Volume 14 Number 1

ACTA CYBERNETICA

Szeged, 1999

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and should be sent in triplicate to any of the Editors. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the József Attila University, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 1999 Numbers 1–2 of Volume 14 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-420-184, Fax:(36)-(62)-420-292.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/local/acta.

H. Bunke
Universität Bern
Institut für Informatik und
angewandte Mathematik
Längass strasse 51., CH-3012 Bern
Switzerland

B. Courcelle
Universitè Bordeaux-1
LaBRI, 351 Cours de la Libèration
33405 TALENCE Cedex
France

J. Demetrovics
MTA SZTAKI
Budapest, P.O.Box 63
H-1502 Hungary

B. Dömölki
IQSOFT
Budapest, Teleki Blanka u. 15-17.
H-1142 Hungary

J. Engelfriet
Leiden University
Computer Science Department
P.O. Box 9512, 2300 RA Leiden
The Netherlands

Z. Ésik
A. József University
Department of Foundations of
Computer Science
Szeged, Aradi vértanúk tere 1.
H-6720 Hungary

L. Lovász
Eötvös Loránd University
Budapest Múzeum krt. 6-8.
H-1088 Hungary

G. Păun
Institute of Mathematics
Romanian Academy
P.O.Box 1-764, RO-70700
Bucuresti, Romania

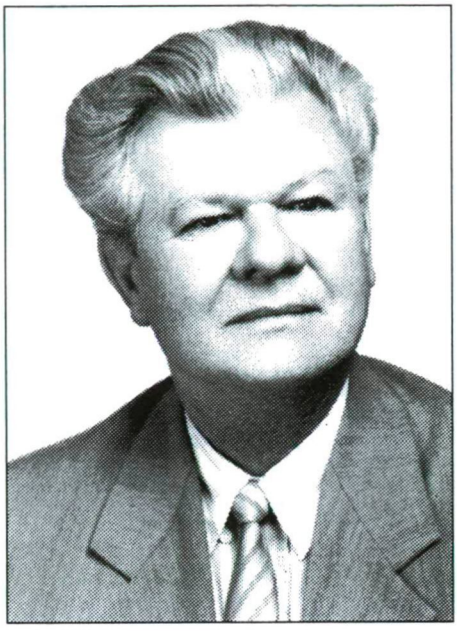
A. Prékopa
Eötvös Loránd University
Budapest, Múzeum krt. 6-8.
H-1088 Hungary

A. Salomaa
University of Turku
Department of Mathematics
SF-20500 Turku 50, Finland

L. Varga
Eötvös Loránd University
Budapest, Múzeum krt. 6-8.
H-1088 Hungary

H. Vogler
Dresden University of Technology
Department of Computer Science
Foundations of Programming
D-01062 Dresden, Germany

G. Wöginger
Technische Universität Graz
Institut für Mathematik (501B)
Steyrergasse 30
A-8010 Graz, Österreich



Professor Ferenc Gécseg

Preface

This issue of Acta Cybernetica contains 15 papers dedicated to Professor Ferenc Gécseg on the occasion of his 60th birthday.

Professor Gécseg has been a very active researcher. His scientific work includes papers on universal algebra, automata, and formal languages. In automata theory, he was mainly interested in various kinds of composition operations on automata. In particular, he introduced a hierarchy of products between the cascade composition and the Glushkov-product, which has extensively been studied by himself and several of his former students and other researchers. More recently, he has extended these investigations to tree automata. His papers and three books (Algebraic Theory of Automata, coauthored by István Peák, Tree Automata, coauthored by Magnus Steinby, and Products of Automata) have greatly influenced the research in automata theory.

Professor Gécseg has contributed a great deal of time to the computer science community by his work as an editor and by organizing several conferences: the automata theory mini-conferences in Szeged in the 1970's, the FCT conference in 1981 and 1989, and the ICALP in 1995. He is the member of the editorial board of several journals, and for almost two decades he was the Editor in Chief of *Acta Cybernetica*. For several years, Professor Gécseg has been Vice-President

of the European Association for Theoretical Computer Science. He has received the highest scientific recognition in Hungary. He is a member of the Hungarian Academy of Sciences and a foreign member of the Finnish Academy of Sciences.

As a teacher, he has been greatly admired by students for his clear presentation of the material. Several of us in Hungary now doing research in theoretical computer science feel very priviliged to have been his former students. We wish Professor Gécseg much success and happiness in the years to come.

Szeged, December 1998.

János Csirik Zoltán Ésik Zoltán Fülöp Balázs Imreh

Acta Cybernetica 14 (1999) 1-12.

On Some Cyclic Connectivity Properties of Directed Graphs (Examples and Problems) *

A. Ádám †

To Professor Ferenc Gécseg on his sixtieth birthday

Introduction

The essence of the paper consists in ten properties (each defining a class of finite directed graphs) listed in § 2 and in open questions (relating to dependencies among the properties) raised in §§ 8–10.

A number of dependence and independence assertions can be deduced easily or follow trivially from the ten properties. The originality of the statements in §§ 3, 8, 9 and of the examples in §§ 6, 7 does not exceed the level of routine consequences of the definitions.

Since the number of properties is ten, one can think "a priori" that the class of graphs which possess at least one property is partitioned into $1023(= 2^{10} - 1)$ subclasses (called types). In fact, the dependency statements imply that there are not more than twenty-one types; on the other hand, examples are got for ten types. The 21 imaginable types correspond in a natural manner to the 21 independent vertex sets of the hierarchy diagram shown in Figure 1. One of the types consists of some connected graphs which are not strongly connected, the remaining ≤ 20 types constitute a partition of the class of the strongly connected graphs.

A part of the open problems concerns to the existence of the eleven types whose non-emptiness is not decided in the article. In the last section an exciting topics is affected: the (fond?) hope for elaborating a structure theory of the strongly connected (directed) graphs.

*Research partially supported by the Hungarian National Foundation for Scientific Research (OTKA) grant, no. T 16389.

†MTA Matematikai Kutatóintézet, H-1364 Budapest, P.O.Box 127., Hungary.

I Notions and facts

§ 1

By a graph, we mean a finite directed graph $G = (V, E)$ where $V(\neq \emptyset)$ is the set of vertices and $E(\neq \emptyset)$ is the set of edges of G. We suppose that G is simple (in detail: any edge joins two *different* vertices and there is at most one edge between any fixed vertex pair) and connected. The outdegree and indegree of $a(\in V)$ are denoted by $\delta^-(a)$ and $\delta^+(a)$, respectively.

Throughout the paper we study graphs whose vertices satisfy[1] $\delta^-(a) \cdot \delta^+(a) \geq 2$; hence transient vertices (i.e., vertices fulfilling $\delta^-(a) = \delta^+(a) = 1$) are excluded.

A graph G is said to be *strongly connected* if, for each ordered vertex pair (a, b), there exists a directed path from a to b.

As usual, we say that two edges $\mathfrak{e} = (a, b)$ and $\mathfrak{f} = (c, d)$ of G are *adjacent* if the number of different elements of the vertex set $\{a, b, c, d\}$ is three. We say that $\mathfrak{e}$ and $\mathfrak{f}$ are *consecutively* adjacent if $b = c$ or $a = d$ holds; $\mathfrak{e}$ and $\mathfrak{f}$ are said to be *oppositely* adjacent if either $a = c$ or $b = d$.

Remark 1 The assumption that transient vertices cannot occur is useful, and it is not a serious restriction. The structure of cycles of a graph is not altered essentially if a transient vertex and the two edges incident to it are replaced by a new edge.

§ 2

By a *cycle* Z of a graph G, we understand a sequence

$$a_1, a_2, \ldots, a_t \tag{2.1}$$

of pairwise different vertices such that $t \geq 3$ and there exist the t (directed) edges

$$(a_1, a_2), (a_2, a_3), \ldots, (a_{t-1}, a_t), (a_t, a_1) \tag{2.2}$$

in G. We say that the vertices in (2.1) and the edges in (2.2) are contained in Z (or, equivalently, that they belong to Z). The cycles (2.1) and $a_2, a_3, \ldots, a_t, a_1$ are considered to equal. We say that the *length* of the cycle (2.1) is t. A cycle of length 3 is also called a *cyclical triangle*.[2]

Let x be an element of either V or E. We say that x is *cyclic* if there is a cycle Z which contains x. Two different elements x, y of $V \cup E$ are called *cyclically completable* if there exists a cycle Z such that both x and y belong to Z.

Next ten conditions (A), (B), ..., (K) will be introduced for a directed graph $G = (V, E)$.

(A) Any vertex of G is cyclic.

(B) Any edge of G is cyclic.

[1] This condition implies that we do not deal with graphs being sheer cycles.

[2] If a_1, a_2, a_3 are pairwise joined and they do not constitute a cyclical triangle, then they form clearly a transitive triangle.

(C) Any pair a, b of vertices of G is cyclically completable.

(D) Any pair $a(\in V), \mathfrak{e}(\in E)$ is cyclically completable.

(E) Any pair $\mathfrak{e}, \mathfrak{f}$ of edges of G is cyclically completable when $\mathfrak{e}$ and $\mathfrak{f}$ are non-adjacent or consecutively adjacent edges.

(F) There exists an $a(\in V)$ such that any pair $a, b(\in V - \{a\})$ is cyclically completable.

(G) There exists an $a(\in V)$ such that any pair $a, \mathfrak{e}(\in E)$ is cyclically completable.

(H) There exists an $\mathfrak{e}(\in E)$ such that any pair $a(\in V), \mathfrak{e}$ is cyclically completable.

(J) There exists an $\mathfrak{e}(\in E)$ such that any pair $\mathfrak{e}, \mathfrak{f}(\in E - \{\mathfrak{e}\})$ is cyclically completable.

(K) There exists an $\mathfrak{e}(\in E)$ such that any pair $\mathfrak{e}, \mathfrak{f}(\in E - \{\mathfrak{e}\})$ is cyclically completable when $\mathfrak{e}$ and $\mathfrak{f}$ are non-adjacent or consecutively adjacent edges.

The class of all (connected simple) graphs G which fulfil (A) is denoted by $\mathbf{A}$. The notations $\mathbf{B}, \mathbf{C}, \ldots, \mathbf{K}$ are used in an analogous sense.

Remark 2 It is well known that Condition (B) is equivalent to the strong connectedness of G.

Remark 3 The condition "any pair of edges is cyclically completable" does not occur among (A)–(K). This condition is not fulfilled by a connected simple graph unless it is a single cycle (cf. Footnote 1).

§ 3

Our aim in the present section is to state thirteen inclusions for the graph classes introduced above. The next ten inclusions follow immediately from how Conditions (A)–(K) have been defined:[3]

$$\mathbf{B} \subseteq \mathbf{A}, \mathbf{C} \subseteq \mathbf{F}, \mathbf{G} \subseteq \mathbf{F}, \mathbf{H} \subseteq \mathbf{F},$$

$$\mathbf{J} \subseteq \mathbf{K}, \mathbf{J} \subseteq \mathbf{G}, \mathbf{E} \subseteq \mathbf{K}, \mathbf{D} \subseteq \mathbf{C}, \mathbf{D} \subseteq \mathbf{G}, \mathbf{D} \subseteq \mathbf{H} \tag{3.1}$$

Lemma 1 *Let $\mathfrak{e} = (b, c)$ be an edge of G such that any pair $\mathfrak{e}, \mathfrak{f}(\in E - \{\mathfrak{e}\})$ is cyclically completable when $\mathfrak{e}, \mathfrak{f}$ are non-adjacent or consecutively adjacent edges. Choose an arbitrary element a of $V - \{b\}$. Then the pair $a, \mathfrak{e}$ is cyclically completable.*

[3]The evident formulae $\mathbf{F} \subseteq \mathbf{A}, \mathbf{D} \subseteq \mathbf{B}, \mathbf{J} \subseteq \mathbf{H}$ are omitted from (3.1). These are consequences of other assertions in this section (by the transitivity of inclusion).

Proof. From among the two possibilities $a = c$ and $a \neq c$ it suffices to treat the second one. There are at least three edges incident to a, we can select an edge $\mathfrak{g}$ out of them such that $\mathfrak{g}$ is either non-adjacent or consecutively adjacent to $\mathfrak{e}$. The pair $\mathfrak{e}, \mathfrak{g}$ is cyclically completable, hence the same holds for the pair $\mathfrak{e}, a$. □

Lemma 1 implies at once

Corollary 1 *We have* $\mathbf{E} \subseteq \mathbf{D}$ *and* $\mathbf{K} \subseteq \mathbf{H}$. □

Proposition 1 *We have* $\mathbf{F} \subseteq \mathbf{B}$.

Proof. Suppose $G \in \mathbf{F}$. Choose an arbitrary edge $\mathfrak{e} = (c, d)$ in G. By (F), a is accessible from d and c is accessible from a. It follows the accessibility of c from d, that is, the cyclicity of $\mathfrak{e}$. (Our idea remains valid even if $a \in \{c, d\}$.) □

II Hierarchy and examples

§ 4

The assertions contained in § 3 (Corollary 1, Proposition 1 and the formulae in (3.1)) determine a hierarchy of the graph classes $\mathbf{A}, \mathbf{B}, \ldots, \mathbf{K}$. This hierarchy is shown in Figure 1.

One can now pose the general problem whether there exists any further interrelation among the ten graph classes or not. A more particular question is whether each of the thirteen inclusions is proper.

In what follows, the general problem is broken up into a number of subproblems. We do not succeed in carrying out a full discussion, a part of the subproblems will be left open. It will be shown, however, that eleven of the thirteen inclusions mentioned above are proper (Proposition 2).

§ 5

Figure 1 shows a cycle-free (directed) graph. There are nine maximal independent[4] vertex sets in this graph:

$$\{\mathbf{A}\}, \{\mathbf{B}\}, \{\mathbf{F}\}, \{\mathbf{C}, \mathbf{J}\}, \{\mathbf{D}, \mathbf{K}\},$$
$$\{\mathbf{D}, \mathbf{J}\}, \{\mathbf{E}, \mathbf{J}\}, \{\mathbf{C}, \mathbf{G}, \mathbf{H}\}, \{\mathbf{C}, \mathbf{G}, \mathbf{K}\}.$$

Thus there are twenty-one non-empty independent vertex sets:

$$\left.\begin{array}{ccccc} \{\mathbf{A}\}, & \{\mathbf{B}\}, & \{\mathbf{C}\}, & \{\mathbf{D}\}, & \{\mathbf{E}\}, \\ \{\mathbf{F}\}, & \{\mathbf{G}\}, & \{\mathbf{H}\}, & \{\mathbf{J}\}, & \{\mathbf{K}\}, \\ \{\mathbf{C}, \mathbf{G}\}, & \{\mathbf{C}, \mathbf{H}\}, & \{\mathbf{C}, \mathbf{J}\}, & \{\mathbf{C}, \mathbf{K}\}, & \\ \{\mathbf{D}, \mathbf{J}\}, & \{\mathbf{D}, \mathbf{K}\}, & \{\mathbf{E}, \mathbf{J}\}, & \{\mathbf{G}, \mathbf{H}\}, & \\ \{\mathbf{G}, \mathbf{K}\}, & \{\mathbf{C}, \mathbf{G}, \mathbf{H}\}, & \{\mathbf{C}, \mathbf{G}, \mathbf{K}\}. & & \end{array}\right\} \quad (5.1)$$

[4] Two vertices of a directed graph $G = (V, E)$ are called independent if they are mutually inaccessible (by directed paths). A subset of V is said to be independent if its elements are pairwise independent. Maximality is understood with respect to set inclusion. It is clear that the independent vertex sets are exactly the subsets of the maximal independent sets. Although the empty set is independent, we shall disregard it.

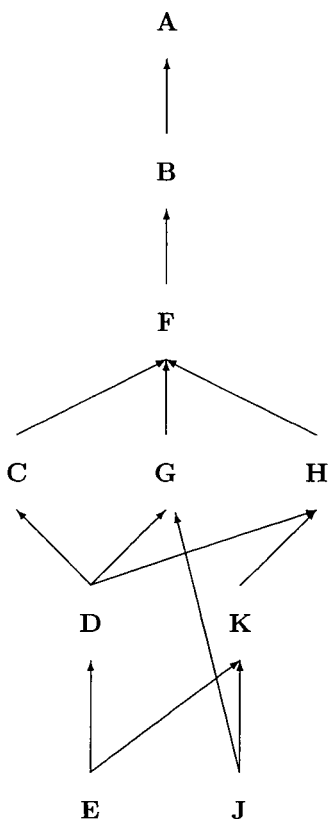


Figure 1

Whenever $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_g\}$ is one of these twenty-one sets (where $1 \leq g \leq 3$), then we define a type of graphs in the following manner:

G is of $(X_1 X_2 \ldots X_g)$-type precisely if

(α) G belongs to $\mathbf{X}_1 \cap \mathbf{X}_2 \cap \ldots \cap \mathbf{X}_g$ and

(β) G does not belong to any of the classes (out of $\mathbf{A}, \mathbf{B}, \ldots, \mathbf{K}$) which are inaccessible from the vertices $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_g$ in the graph of Figure 1.

For example, G is contained in the (CJ)-type exactly when
G belongs to $\mathbf{C} \cap \mathbf{J}$ (hence to $\mathbf{A}, \mathbf{B}, \mathbf{F}, \mathbf{G}, \mathbf{H}, \mathbf{K}$ also), and
G does not belong to $\mathbf{D}$ (thus also $G \notin \mathbf{E}$).

In sense of this definition, the class of (connected) graphs fulfilling Condition (A) is partitioned into at most twenty-one types.[5] The remaining two sections of Chapter II are devoted to giving examples which show the non-emptiness of ten types.

§ 6

We examine in this section some strongly connected graphs (without transient vertices) which have four or five vertices. The outdegrees and indegrees of the vertices of a graph are expressed by a matrix of form

$$\begin{pmatrix} \delta^-(a_1) & \delta^-(a_2) & \ldots & \delta^-(a_v) \\ \delta^+(a_1) & \delta^+(a_2) & \ldots & \delta^+(a_v) \end{pmatrix}$$

where the vertices are numbered in such a manner that

(i) $\delta^-(a_1) \geq \delta^-(a_2) \geq \ldots \geq \delta^-(a_v)$ (where $v = |V|$), and

(ii) $1 \leq i < v$, $\delta^-(a_i) = \delta^-(a_{i+1})$ imply $\delta^+(a_i) \geq \delta^+(a_{i+1})$.

Example 1 There is only one strongly connected tournament with four vertices (apart from isomorphy), see Figure 2/a. The degree matrix of this graph is

$$\begin{pmatrix} 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 \end{pmatrix}.$$

There is one cycle of length four: *abcd* and there are two cyclical triangles: *abd* and *acd*.

The fulfilment of Condition (C) is clear. (J) is satisfied with the edge (da). (D) does not hold since the vertex b is not cyclically completable with the edge (ac).

We have got that the type of this graph is (CJ).

In the following five examples, graphs with five vertices and eight edges are considered. The degree matrix of Examples 2–5 is

$$\begin{pmatrix} 2 & 2 & 2 & 1 & 1 \\ 2 & 1 & 1 & 2 & 2 \end{pmatrix}.$$

[5] By (β), the subclasses called types are pairwise disjoint. We used the words "at most" because it is not sure that all the types are non-empty.

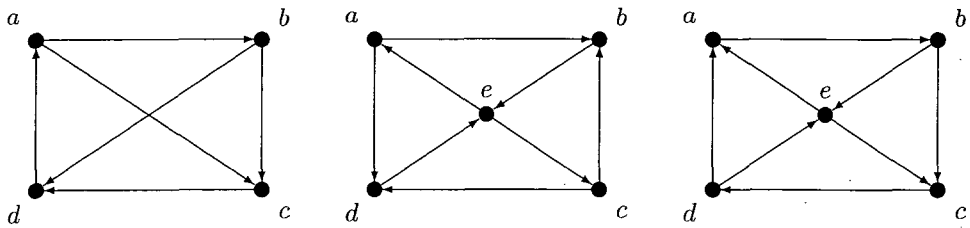


Figure 2

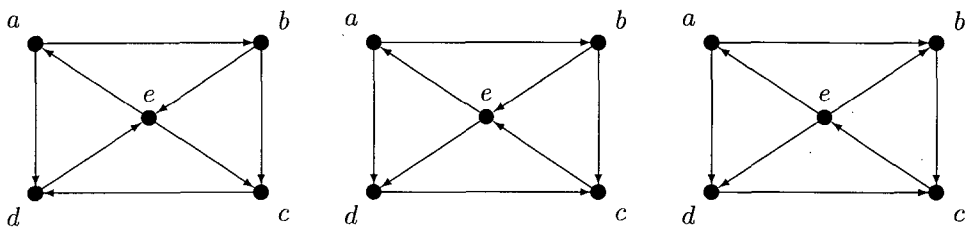


Figure 3

Example 2 In the graph of Figure 2/b there are four cyclic triangles: abe, cbe, cde and ade. There is no cycle of length four or five. Condition (G) is satisfied with e. The graph fulfils neither (C) nor (H). ((C) is false since with a and c form a pair which is not cyclically completable.)

The type of this graph is (G).

Example 3 The graph of Figure 2/c has five cycles; in detail, $abcde$ and $abecd$ are ones of length five, the length of $abcd$ is four, furthermore, abe and cde are cyclical triangles. (J) is satisfied with the edge (ab), the fulfilment of (E) can be checked easily.

The type of this graph is (EJ). It is contained in each of the classes $\mathbf{A}, \mathbf{B}, \ldots, \mathbf{K}$.

Example 4 The length of any cycle of the graph of Figure 3/a is five or three. There is one cycle containing all vertices: $abcde$, and there are three cyclical triangles: abe, ade and cde. Conditions (G), (K) are satisfied with $e, (de)$, respectively; the fulfilment of (C) is clear. (D) is false for this graph (for example, a and (ec) are not cyclically completable). The falsity of (J) follows from the fact that, for an arbitrary edge $\mathfrak{f}$, there is an edge $\mathfrak{g}$ such that $\mathfrak{f}, \mathfrak{g}$ are oppositely adjacent edges.

This graph is of type (CGK).

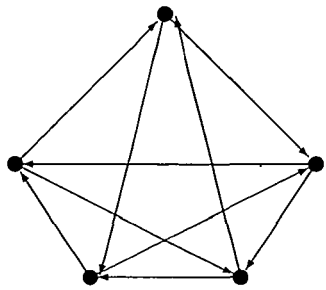


Figure 4

Example 5 Any cycle of the graph of Figure 3/b is of length three or four. There are two cycles whose length is four: *abce* and *adce*; there are two cyclical triangles: *abe* and *ced*. (G) is satisfied with e, (K) is fulfilled with (ce). (C) is not valid because b and d are not cyclically completable. (J) does not hold (by the same reason as in the preceding example)!

The type of this graph is (GK).

Example 6 The degree matrix of the graph of Figure 3/c is

$$\begin{pmatrix} 3 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 \end{pmatrix}.$$

There is no cycle of length five. Two cycles have length four: *abce* and *adce*; there are two cyclical triangles: *bce* and *ced*. Condition (J) holds with (ce). Condition (C) is not fulfilled since b and d are not cyclically completable.

The type of this graph is (J).

Example 7 Consider the graph of Figure 4. Each outdegree and indegree equals 2 in it. It is easy to see (without a detailed examination of the cycles) that this highly symmetric tournament satisfies (E). Condition (J) is not fulfilled (similarly to Examples 4 and 5).

We have got that this graph belongs to the type (E).

Remark 4 It seems that the remaining strongly connected graphs having five vertices (each non-transient) do not represent any other type than the types to which Examples 1–7 belong. For the reader who wants\to recapitulate the investigation of these graphs, the following method can be advised:

first to get an overview of the possible degree matrices,

for any degree matrix, to construct all of its graph realizations,

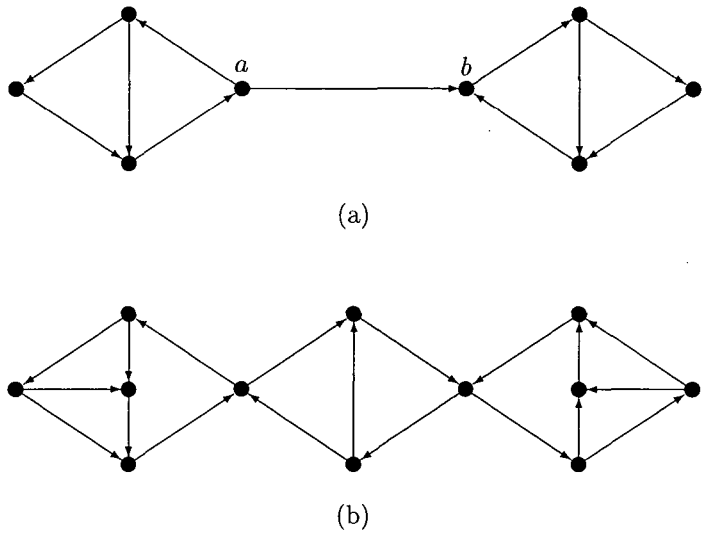


Figure 5

if two graphs are dual[6] to each other, then to examine only one of them (because duality does not alter the structure of cycles essentially).

§ 7

We have seen in § 6 some examples which were obtained by scanning all graphs with a very small number of vertices, they show the non-emptiness of seven types. In the present section examples for three additional types will be given.[7]

The fact that the type (A) of graphs is not empty is clear (it is well known that the class of strongly connected graphs does not exhaust the class of connected graphs in which every vertex is cyclic). For the sake of completeness of the treatment, we put first an instance for this type.

Example 8 Every vertex of the graph in Figure 5/a is cyclic, and (ab) is a non-cyclic edge. Thus the graph belongs to the type (A).

Example 9 Figure 5/b shows a graph for which (B) is valid (or, equivalently, it is strongly connected), but (F) does not hold. This means that the graph is of type (B).

[6]Two directed graphs are said to be dual if one is obtained from the other by reversing the orientation of all the edges. It may happen that a graph is isomorphic to its dual.

[7]In the course of constructing these graphs, the restriction that transient vertices are forbidden needed to be regarded.

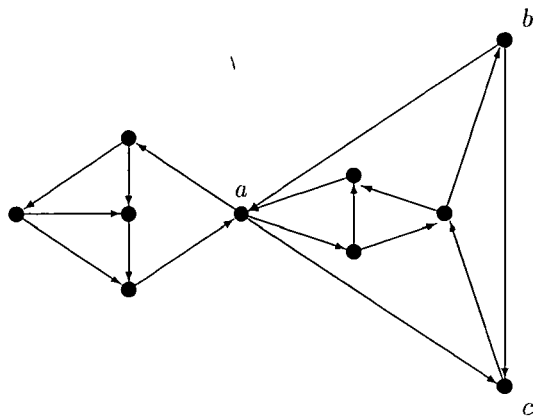


Figure 6

Example 10 The graph in Figure 6 satisfies (F) (with the vertex a). It is easy to see that (C) and (H) are not valid. (G) is not fulfilled also (observe that a and the edge (bc) are not cyclically completable). Therefore this graph is of type (F).

III Overview and open questions

§ 8

Remember that twenty-one graph types have been introduced in § 5. We have seen in Sections 6–7 examples for ten types. The question of the existence of the remaining types is open:

Problem 1 *Decide for any of the eleven types (C), (D), (H), (K), (CG), (CH), (CK), (DJ), (DK), (GH), (CGH) – consisting of directed graphs – whether the type is empty or not.*

Problem 1 is a comprehensive question in the sense that the subsequent Problems 2 and 3 are (essentially) particular cases of it.

Recall the hierarchy shown in Figure 1. We are going to suggest some possibilities for improving this hierarchy. Next we consider the pairs of graph classes (out of $\mathbf{A}, \mathbf{B}, \ldots, \mathbf{K}$) for which the corresponding vertices are adjacent in Figure 1.

Proposition 2 *We have the proper inclusions*

$$\mathbf{B} \subset \mathbf{A}, \mathbf{F} \subset \mathbf{B}, \mathbf{C} \subset \mathbf{F}, \mathbf{G} \subset \mathbf{F},$$

$$\mathbf{H} \subset \mathbf{F}, \mathbf{D} \subset \mathbf{C}, \mathbf{D} \subset \mathbf{G}, \mathbf{D} \subset \mathbf{H},$$

$$\mathbf{E} \subset \mathbf{K}, \mathbf{J} \subset \mathbf{G}, \mathbf{J} \subset \mathbf{K}.$$

Proof. Examples 8, 9 show that $\mathbf{B}$ is properly included in $\mathbf{A}$ and $\mathbf{F}$ is properly included in $\mathbf{B}$, respectively. The inclusion $\mathbf{C} \subset \mathbf{F}$ is guaranteed by[8] Example 2. The remaining eight inclusions are ensured by Examples 10, 2, 1, 1, 1, 1, 2, 4, respectively. □

We do not have examples for the strictness of the inclusions $\mathbf{K} \subseteq \mathbf{H}$ and $\mathbf{E} \subseteq \mathbf{D}$, hence we can raise:

Problem 2 *Decide the validity of the equalities* $\mathbf{K} = \mathbf{H}$ *and* $\mathbf{E} = \mathbf{D}$.

§ 9

There are nine independent vertex pairs in the graph of Figure 1. Our present aim is to discuss the pairs of graph classes which correspond to these vertex pairs.

Examples 3, 6 and 7 guarantee the truth of the following assertions:

Proposition 3 *The intersection* $\mathbf{E} \cap \mathbf{J}$ *is not empty and it is properly included both by* $\mathbf{E}$ *and by* $\mathbf{J}$. *The intersections* $\mathbf{D} \cap \mathbf{J}$ *and* $\mathbf{C} \cap \mathbf{J}$ *are proper in the same sense.* □

Any of the formulae

$$\mathbf{G} \subseteq \mathbf{K}, \mathbf{G} \subseteq \mathbf{H}, \mathbf{G} \subseteq \mathbf{C}, \mathbf{H} \subseteq \mathbf{C}, \mathbf{K} \subseteq \mathbf{C}, \mathbf{K} \subseteq \mathbf{D},$$

is disproved either by Example 2 or by Example 6; our examples leave open the truth of the (strict) inclusions in the opposite sense. Thus we can pose:

Problem 3 *Decide the validity of the inclusions*

$$\mathbf{K} \subset \mathbf{G}, \mathbf{H} \subset \mathbf{G}, \mathbf{C} \subset \mathbf{G}, \mathbf{C} \subset \mathbf{H}, \mathbf{C} \subset \mathbf{K}, \mathbf{D} \subset \mathbf{K}.$$

§ 10

We have throughout adopted in the above considerations that transient vertices are forbidden. After determining the hierarchy completely (i.e., after solving Problem 1), the question may arise how the hierarchy changes when transient vertices are allowed. The solution of this question does not seem to be difficult.

Now we turn to another possibility of varying the subject. In our former analysis, graphs having cut vertices[9] were not excluded. Cut vertices have occured in Examples 8, 9, 10 actually. It is evident that Example 8 can be replaced by a twofold connected graph; the same is, however, not trivial for Examples 9 and 10. Consequently we formulate

Problem 4 *Is the hierarchy of the graph classes* $\mathbf{B}, \mathbf{C}, \ldots, \mathbf{K}$ *modified if we restrict ourselves to graphs without cut vertices? Especially, do the formulae* $\mathbf{F} \subset \mathbf{B}$ *and* $\mathbf{G} \subset \mathbf{F}$ *remain valid after this restriction?*

[8] Instead of Example 2, any of Examples 5, 6, 10 is suitable for this purpose. In what follows, we mention only one instance in analogous situations, and the search for other appropriate examples is left to the reader.

[9] In other words, exactly onefold connected graphs. The orientation of the edges is indifferent in this notion.

§ 11

In the hierarchy of graphs studied by us, the intersection $\mathbf{E} \cap \mathbf{J}$ is the common part of all classes. It can be hoped that a structural description of this relatively narrow graph class will be elaborated:

Problem 5 *Characterize the structure of the graphs which belong to the type (EJ).*

A bold challenge is initiated in the last open question of the paper. A complete elucidation of this question would imply a systematization of the structure of all finite directed graphs. Even if this goal will not prove to be successful, partial solutions are of importance also:

Problem 6 *Consider a graph class* $\mathbf{X}$ *from among the nine classes* $\mathbf{B}, \mathbf{C}, \ldots, \mathbf{K}$. *Let a constructive procedure be obtained how the members of the class* $\mathbf{X}$ *can be produced from graphs belonging to classes which are proper subclasses of* $\mathbf{X}$ *(in the hierarchy).*

Remark 5 It is at once clear that the solution of Problem 6 for all the nine classes yields the structural overview of the *strongly connected* graphs. This observation can be supplemented by two well-known facts:

(i) any directed graph G admits a unique decomposition into a cycle-free graph so that each maximal strongly connected subgraph of G is contracted into a single vertex;

(ii) it is possible to get a good survey of the structure of cycle-free (directed) graphs.

Details may be found in Chapters 3 and 10 of the book [1] of Harary, Norman, and Cartwright.

References

[1] Harary, F., Norman, R. Z., and Cartwright, D., *Structural models: An introduction to the theory of directed graphs,* Wiley, New York, 1965. (French translation: Dunod, Paris, 1968.)

Acta Cybernetica **14** (1999) 13-25.

Two simple algorithms for bin covering

J. Csirik * J. B. G. Frenk † M. Labbé ‡ S. Zhang §

Dedicated to Professor Ferenc Gécseg on the occasion of his 60th birthday

Abstract

We define two simple algorithms for the bin covering problem and give their asymptotic performance.

1 Introduction

In this chapter we consider the following version of bin packing sometimes called *dual bin packing* or *bin covering* : given a list

$$L = (a_1, a_2, \ldots, a_n)$$

of items with size $s(a_i)$ for each item a_i, and a bin capacity C,

$$C > \max_{1 \leq i \leq n} s(a_i),$$

pack the elements of L into a maximum number of bins so that the sum of sizes in any bin is at least C. This means, that we have to fill as many bins as possible. It is clear, that we can normalize the problem so that C is equal to 1 and $s(a_i) < 1$ for every $1 \leq i \leq n$. The above problem was investigated for the first time by Assmann (cf.[1]) and Assmann *et al.*(cf.[2]). In particular, they showed that the problem is NP-hard. Furthermore, they provided the first approximation algorithms and proved their worst-case performance. Some average-case analysis was also performed.

We denote by $OPT(L)$ the optimal, i.e. the maximal number of filled bins for a list

$$L = (a_1, a_2, \ldots, a_n)$$

*Department of Informatics, József Attila University, Árpád tér 2, H-6720 Szeged, Hungary

†Erasmus University of Rotterdam, Faculty of Economics, Postbus 1738, 3000 DR Rotterdam, Netherlands

‡Université Libre de Bruxelles, Mathematiques du Triomphe, 1050 Bruxelles, Belgium

§Erasmus University of Rotterdam, Faculty of Economics, Postbus 1738, 3000 DR Rotterdam, Netherlands

and we define for every $k \geq 1$

$$R_A(k) := \min \left\{ \frac{A(L)}{k} \mid OPT(L) = k \right\}, \tag{1}$$

where $A(L)$ denotes the number of bins filled by algorithm A applied to the list L. The *performance ratio* or *asymptotic worst case ratio* of A, denoted by R_A, is now given by

$$R_A := \liminf_{k \to \infty} R_A(k). \tag{2}$$

Clearly, $R_A(k) \leq 1$ for every $k \geq 1$, and hence $R_A \leq 1$.

For an equivalent definition of R_A we observe that $R_A \geq K_1$ if there exist two constants K_1 and K_2 such that

$$A(L) \geq K_1 \cdot OPT(L) + K_2 \tag{3}$$

for every list L. Clearly the largest possible K_1 satisfying this inequality equals R_A. By this definition it is obvious that a heuristic A_1 is at least as good as heuristic A_2 (from a worst case point of view) if $R_{A_1} \geq R_{A_2}$.

2 Preliminary results

It is very natural to adapt classical bin packing heuristics to the dual problem. So the first heuristic is **Dual Next Fit** (DNF).

1. Put the first element into the first bin.
2. While there is an unpacked item, do the following: Let a_i be the first unpacked element, and let B_j be the bin that is not yet filled (if all bins are filled, we take a new empty bin as B_j). Place a_i in bin B_j.

This algorithm has the nice property that it uses only one bin at a time and that it works on-line. The algorithm runs in $O(n)$ time. However, the asymptotic worst case bound of DNF is not very good (cf.[2]).

Lemma 1 $R_{DNF} = 1/2$.

From the 'bad' lists for this algorithm it follows, that - contrary to classical bin packing - sorting the items in nonincreasing order does *not* improve the performance of the heuristic. This implies, that the heuristic **Next Fit Decreasing** (NFD) of classical bin packing adapted to the dual bin packing problem also has a performance ratio of $1/2$.

The next idea is to use First Fit type heuristics instead of Next Fit, i.e. to use all opened bins instead of the last one. However, it has no meaning in this case, because after filling a bin, it is useless to place further items into it. That means that neither **First Fit** nor **First Fit Decreasing** adapted to the dual bin packing problem have a larger performance ratio than Next Fit.

An improvement of the performance ratio of 1/2 was achieved by Assmann *et al.*(cf.[2]) by defining an *artificial upper bound* on the sum of sizes of elements placed into the same bin. This upper bound can be regarded as the capacity of a bin and leads to some similarity with classical bin packing. However, after packing the items by a good heuristic for the classical bin packing problem, it might happen that in some of the bins the sum of item sizes is less than 1. Hence we can use a second step to fill these bins. The algorithm based on the above observation, proposed by Assman *et al.*, is called **First Fit Decreasing with parameter r** (FFD_r) and proceeds as follows:

Let $1 < r < 2$.

Phase I. ("Classical FFD")

1. Presort the items in L so that
$$s(a_1) \geq s(a_2) \geq \ldots \geq s(a_n).$$
2. While there is still an unpacked element, do the following: Let a_i be the first unpacked item and let B_j be the first (leftmost) unfilled bin with a current total content smaller than or equal to $r - s(a_i)$. If such a bin exists, place a_i in B_j, otherwise open a new empty bin and pack a_i into this bin.

Phase II. (Repacking unfilled bins)

1. While there is more than one open nonfilled bin, remove an item from the rightmost such bin and add it to the leftmost one.

The time complexity of FFD_r can be seen to be $O(n \log n)$. For this algorithm the following result holds (cf.[2]).

Lemma 2 $R_{FFD_r} = 2/3 \quad for \quad 4/3 < r < 3/2.$

Assmann et al. also suggested a further improvement by defining a really sophisticated algorithm, called **Iterated Lowest Fit Decreasing** ($ILFD$). To define this heuristic we consider first the following problem: Given the list L and a fixed number N of bins, what is the *maximum* possible value for the minimum bin level in a packing of L into N bins? From a good heuristic A for this problem we can derive a good approximation algorithm for the bin covering problem by iteratively applying this algorithm A. We denote by $A(L, N)$ the minimum bin level in the packing of L generated by the heuristic A if the number of bins is fixed by N. Now the algorithm iteratively applying A proceeds as follows:
ITERATED "A"

1. Let $UB = \sum_{i=1}^{n} s(a_i)$, $LB = 1$. (Clearly $LB \leq OPT(L) \leq UB$.)
2. While $UB - LB > 1$ take $N = \lfloor (LB + UB)/2 \rfloor$ and apply heuristic A. If $A(L, N) > 1$ take $LB = N$, otherwise $UB = N$.

The resulting algorithm gives a feasible solution of the dual bin packing problem with LB bins.

Clearly, the performance of this method depends on the choice of A. While the problem to be solved by A is closely related to multiprocessor scheduling problems, it seems natural to use for the heuristic A the **Lowest Fit Decreasing** (LFD) algorithm, as studied by Graham (cf.[4]) and Deyermeyer *et al.*(cf.[3]). This algorithm proceeds as follows:

1. Order L so that $s(a_1) \geq s(a_2) \geq \ldots \geq s(a_n)$ and start with N empty bins.
2. While there is an unpacked item in L do the following: let a_i be the first unpacked item and let B_j be the bin with minimum level (in case of ties, choose the rightmost). Put a_i into B_j.

It is not difficult to verify that the time complexity of $ILFD$ is $O(n \log^2 n)$. Furthermore, one can prove the following result (cf.[2]).

Lemma 3 $R_{ILFD} = 3/4$.

3 Two new simple algorithms

Now we will show that the same performance bounds can also be achieved by simpler algorithms too. First we discuss the heuristic **Simple** (SI). This algorithm proceeds as follows:

1. Sort the items of list L into nonincreasing order, i.e. from now on we assume that
$$s(a_1) \geq s(a_2) \geq \ldots \geq s(a_n).$$
2. Let k_1 denote the index satisfying
$$\sum_{i=1}^{k_1} s(a_i) < 1 \quad \text{and} \quad \sum_{i=1}^{k_1+1} s(a_i) \geq 1.$$
Pack the elements $a_1, a_2, \ldots, a_{k_1}$ into the first bin. Fill the remaining space in the bin with items from the end of the list, i.e. with $a_n, a_{n-1}, \ldots$ untill the sum of sizes of items in the bin is at least equal to one and remove the packed items from the list.
3. Renumber the indices of the remaining items and repeat step 2 untill the list is empty.

Lemma 4 *For all lists L,*
$$SI(L) \geq \frac{2}{3} \cdot OPT(L) - \frac{2}{3}.$$

Proof. Let us assume that the last item in the last filled bin is a_{last} and distinguish the cases $s(a_{\text{last}}) \leq 1/2$ and $s(a_{\text{last}}) > 1/2$.

If $s(a_{\text{last}}) \leq 1/2$ then the total sum of item sizes in the last filled bin is bounded above by $1 + s(a_{\text{last}}) \leq 3/2$. Moreover, since all the last items in the remaining filled bins are by the definition of SI always smaller than or equal to $s(a_{\text{last}})$ we obtain that the total sum of item sizes of all the filled bins is bounded above by 3/2. As we have at most one non-filled bin, this implies

$$3/2 \cdot SI(L) + 1 \geq s(L).$$

Since $s(L) \geq OPT(L)$ we obtain

$$SI(L) \geq \frac{2}{3}(OPT(L) - 1)$$

for all lists L and so the lemma holds in this case.

If $s(a_{\text{last}}) > 1/2$, we only consider the case where all the opened bins are filled. If this does not hold (i.e. we have one non-filled bin) the proof can be easily adapted. Now it is clear that the SI-packing has the following structure:

a_{n_1}	a_{n_2}	a_{n_3}	$\cdots$	a_{n_k}				
$\vdots$	$\vdots$	$\vdots$		$\vdots$				
a_n	a_{n_1-1}	a_{n_2-1}		$a_{n_{k-1}-1}$	a_{n_k-1}	a_{n_k-2}		$a_{n_k-SI(L)+k}$
a_1	a_2	a_3		a_k	a_{k+1}	a_{k+2}	$\cdots$	$a_{SI(L)}$
B_1	B_2	B_3	$\cdots$	B_k	B_{k+1}	B_{k+2}		$B_{SI(L)}$

where $a_{n_k-SI(L)+k} = a_{\text{last}}$, $n_k - SI(L) + k = SI(L) + 1$, $s(a_{n_k-SI(L)+k}) > 1/2$ and $n > n_1 > n_2 > \ldots > n_k$.

Call the elements $a_n, a_{n-1}, \ldots, a_{n_1+1}, a_{n_1-1}, \ldots, a_{n_2+1}, \ldots, a_{n_{k-1}-1}, \ldots, a_{n_k+1}$ type-A items and the remaining (i.e. the first and the last element in each bin) type-B items, and consider the optimal packing. Define

$k_A :=$ # bins in the optimal packing with only type-A items,
$k_{AB} :=$ # bins in the optimal packing with exactly one type-B item,
$k_{BB} :=$ # bins in the optimal packing with more than one type-B items.

Clearly

$$OPT(L) = k_A + k_{AB} + k_{BB}. \tag{4}$$

Moreover, by the definition of SI we observe that

$$\sum_{i \in type-A} s(a_i) + \sum_{i=1}^{k} s(a_i) < k$$

and this implies, since $a_1, a_2, \ldots, a_k$ are the biggest B-items, that $k_{AB} < k$. Applying the above inequality again and using $s(a_i) > 1/2$, for every $i \leq k$, we get:

$$k_A + k_{AB} \leq \sum_{i \in type-A} s(a_i) + \sum_{i=1}^{k_{AB}} s(a_i) <$$

$$< \frac{k + k_{AB}}{2} \leq \frac{SI(L) + k_{AB}}{2}. \tag{5}$$

Finally, in order to obtain an upper bound for k_{BB} we note that the total number of type-B items in bins with more than one type-B item is given by $2 \cdot SI(L) - k_{AB}$ and hence

$$k_{BB} \leq \frac{2 \cdot SI(L) - k_{AB}}{2}. \tag{6}$$

By (4) and the upper bounds in (5) and (6) we obtain

$$OPT(L) \leq \frac{SI(L) + k_{AB}}{2} + \frac{2 \cdot SI(L) - k_{AB}}{2} = 3/2 \cdot SI(L).$$

and hence the desired result is proved.

□

In the above lemma we proved that $R_{SI} \geq 2/3$. Furthermore, this lower bound can be achieved, as will be shown by the next result.

Theorem 5 $R_{SI} = 2/3$.

Proof. Let us define the lists $L_n, n \geq 1$ by the sizes of the elements and consider

$$L_n = \left(\frac{3}{4}, \underbrace{\frac{1}{2} - \varepsilon, \frac{1}{2} - \varepsilon, \ldots, \frac{1}{2} - \varepsilon}_{6n+1 \text{ times}}, \underbrace{2\varepsilon, 2\varepsilon, \ldots, 2\varepsilon}_{3n \text{ times}} \right).$$

If

$$\varepsilon < \frac{1}{24n}$$

then

$$OPT(L_n) = 3n + 1$$

and

$$SI(L_n) = 2n + 1.$$

Hence we obtain

$$\frac{SI(L_n)}{OPT(L_n)} = \frac{2}{3} + \frac{1}{9n + 3}$$

and this implies $\lim_{n\to\infty} \frac{SI(L_n)}{OPT(L_n)} = \frac{2}{3}$. Applying Lemma 4 finally yields the desired result.

□

It is easy to see that similarly we can characterize the performance of the heuristic SI if $s(a_i) \leq 1/k$ for all a_i, where k is some positive integer. For this case the next result holds.

Theorem 6 *If $s(a_i) \leq 1/k$ for all items in $L = (a_1, a_2, \ldots, a_n)$, where k is a positive integer, then the worst case performance of the heuristic SI is at least $\frac{k+1}{k+2}$.*

Proof. The proof directly follows that of Lemma 4 .

Finally, we consider an improved version of the SI-heuristic. Before introducing this so called **Improved simple** heuristic (ISI) we divide the list L into the following three parts.

i)		$s(x_1) \geq s(x_2) \geq s(x_p) \geq 1/2$	(X-sublist)
ii)	$1/2 >$	$s(y_1) \geq s(y_2) \geq \ldots \geq s(y_r) \geq 1/3$	(Y-sublist)
iii)	$1/3 >$	$s(z_1) \geq s(z_2) \geq \ldots \geq s(z_m)$	(Z-sublist)

Clearly $p + r + m = n$. Now the ISI-heuristic is defined as follows.

Phase 1. If $s(x_1) \geq s(y_1) + s(y_2)$, then pack x_1 into an empty bin, otherwise pack y_1 and y_2 into an empty bin. Fill the just opened bin with elements from the end of the Z-sublist, i.e. with $z_m, z_{m-1}, \ldots$ untill the bin is filled. Remove the packed elements from the corresponding sublists and repeat packing untill either $X \cup Y$ or Z is empty.

Phase 2. If after phase 1, $X \cup Y$ is empty, pack the remaining elements in the Z-sublist according to the Next-Fit heuristic. Otherwise, if Z is empty, pack the remaining x-elements by two in a bin and the remaining y-elements by three.

For the above heuristic the next result holds.

Lemma 7 *The worst case performance ratio of the heuristic ISI is at least equal to 3/4.*

Proof. To verify the above result it is sufficient to prove that $ISI(L) \geq 3/4(OPT(L) - 4)$ for all lists L. This is easy if $X \cup Y$ is empty after phase 1. Observe that in this case the last elements of all filled bins (after the execution of the heuristic) are elements from the Z-sublist and hence the sum of sizes in each filled bin is bounded from above by 4/3. By an argument similar to that used in the first part of Lemma 4 the desired inequality follows.

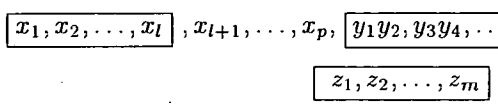


Figure 1: The packing by Improved Simple.

Subsequently we consider the case where the Z-sublist is empty after phase 1 and assume that $x_1, x_2, \ldots, x_l$ from the X-sublist and $y_1y_2, y_3y_4, \ldots, y_{2k-1}y_{2k}$ from the Y-sublist are used in the first phase (cf.Figure 1).

By the definition of the heuristic we obtain

$$ISI(L) \geq l + k + \left\lfloor \frac{p-l}{2} \right\rfloor + \left\lfloor \frac{r-2k}{3} \right\rfloor - 1 \geq \frac{l}{2} + \frac{k}{3} + \frac{p}{2} + \frac{r}{3} - 3. \tag{7}$$

We now have to derive an upper bound on $OPT(L)$. In order to do so call the elements of the Z-sublist, which were packed last in a filled bin by the ISI-heuristic, B-items, and the remaining z-elements A-items and consider the optimal packing. We now rename A-items A^*-items if these A-items in the optimal packing are packed either with only A-items or with one or two y-elements or with exactly one x-element and define

k_{A^*} := # bins in the optimal packing with only A^* items,

k_{A^*X} := # bins in the optimal packing with only A^* items and exactly one element from the X-sublist,

k_{A^*Y} := # bins in the optimal packing with only A^* items and exactly one element from the Y-sublist,

k_{A^*YY} := # bins in the optimal packing with only A^* items and exactly two elements from the Y-sublist,

k_{others} := # of other bins in the optimal packing.

Clearly,

$$OPT(L) = k_{A^*} + k_{A^*X} + k_{A^*Y} + k_{A^*YY} + k_{others}. \tag{8}$$

To obtain an upper bound on the first four terms in (8) we note by the construction of the ISI-packing that

$$\sum_{i \in type-A^*} s(a_i) + \sum_{i=1}^{l} s(x_i) + \sum_{i=1}^{2k} s(y_i) < l + k \tag{9}$$

Furthermore, if $k_{A^*} + k_{A^*X} + k_{A^*Y} + k_{A^*YY} > l + k$, it is possible by the feasibility of the optimal packing and the definition of A^* items to pack more than $l + k$ bins in the first phase of the ISI heuristic and since this does not hold we get

$$k_{A^*} + k_{A^*X} + k_{A^*Y} + k_{A^*YY} \leq l + k. \tag{10}$$

To simplify notations, we now define $l' := k_{A^*X}$ and $2k' := k_{A^*Y} + 2k_{A^*YY}$. Then, by (10), it follows immediately that

$$l' + k' \leq k_{A^*} + k_{A^*X} + k_{A^*Y} + k_{A^*YY} \leq l + k. \tag{11}$$

It turns out that the inequality derived in (10) can be improved as follows. Clearly,

$$k_{A^*} + k_{A^*X} + k_{A^*Y} + k_{A^*YY} \leq \sum_{i \in type-A^*} s(a_i) + \sum_{i=1}^{l'} s(x_i) + \sum_{i=1}^{2k'} s(y_i). \tag{12}$$

By (9), it follows that the upper bound in (12) can be bounded from above by

$$U := l + k + \sum_{i=1}^{l'} s(x_i) - \sum_{i=1}^{l} s(x_i) + \sum_{i=1}^{2k'} s(y_i) - \sum_{i=1}^{2k} s(y_i). \tag{13}$$

In order to bound U we distinguish the following four cases:

i) $l \geq l'$ and $2k \geq 2k'$,
ii) $l < l'$ and $2k \geq 2k'$,
iii) $l \geq l'$ and $2k < 2k'$,
iv) $l < l'$ and $2k < 2k'$.

By (11) case (iv) will never occur and hence we only have to consider i), ii) and iii).

Clearly, if i) holds,

$$U = l + k - \sum_{i=l'+1}^{l} s(x_i) - \sum_{i=2k'+1}^{2k} s(y_i). \tag{14}$$

By the definition of ISI (cf.Figure 1), we now observe that

$$s(x_i) \geq s(x_l) \geq s(y_{2k+1}) + s(y_{2k+2}) > \frac{2}{3},$$

for every $i \leq l$, if $r \geq 2k + 2$.
Moreover, if $r \leq 2k + 1$, then $s(x_l)$ might be smaller than $2/3$ and hence the remaining elements in the X-sublist after phase 1 are always smaller than $2/3$. This observation implies that in phase 2 the sum of the sizes in a filled bin is bounded above by $4/3$ and together with the argument that in phase 1 the sum of sizes in a filled bin are also bounded above by $4/3$ the desired inequality follows. By this observation we may therefore assume that $s(x_i) \geq 2/3$ for every $i \leq l$ and this yields, by (14),

$$U \leq l + k - \frac{2}{3}(l - l') - \frac{1}{3}(2k - 2k') = \frac{1}{3}(l + k) + \frac{2}{3}(l' + k') \tag{15}$$

If ii) holds we obtain by (13) that

$$U = l + k + \sum_{i=l+1}^{l'} s(x_i) - \sum_{i=2k'+1}^{2k} s(y_i). \tag{16}$$

Observe by the definition of the first phase of the ISI-heuristic (cf.Figure 1) that the sum of sizes of any two y_i elements, $i = 2k'+1, \ldots, 2k$ is always bigger than the size of any x_i-element, $i = l+1, \ldots, l'$. This implies

$$\sum_{i=l+1}^{l'} s(x_i) \leq \sum_{i=2k'+1}^{2k'+2l'-2l} s(y_i)$$

and since by (11) $2k' + 2l' - 2l \leq 2k$, we obtain

$$\begin{aligned} U &\leq l + k - \sum_{i=2k'+2l'-2l+1}^{2k} s(y_i) \leq l + k - \frac{1}{3}(2k - (2k' + 2l' - 2l)) \\ &= \frac{l+k}{3} + \frac{2}{3}(l' + k') \end{aligned} \tag{17}$$

which is the same as inequality (15).

In order to bound U by the same upper bound as in (17), when iii) holds, we use a similar argument, i.e. we replace the remaining y-items by an x-item and using the lower bound for x_i yields the desired result.

Hence we have proved that in all cases the improved upper bound

$$U \leq \frac{l+k}{3} + \frac{2}{3}(l' + k') \tag{18}$$

holds.

By (8), we also need an upper bound on k_{others}. These remaining k_{others} bins contain $p - l'$ x-items, $r - 2k'$ y-items, $l + k$ B-items and some A-items. By the definition of k_A, these A-items are always contained in a bin with some of the above elements and hence do not count in the computation of an upper bound for k_{others}. For this upper bound computation we consider four subcases.

A) $l + k \geq p - l' + \frac{r-2k'}{2}$.

If this holds, the best we can hope for is to pack one X-element with one B-item, two Y-elements with one B-item and the remaining B-items by four. Hence

$$k_{others} \leq p - l' + \frac{r - 2k'}{2} + \frac{l + k - (p - l' + \frac{r-2k'}{2})}{4} \tag{19}$$

and by (18) and (8) this implies

$$OPT(L) \leq \frac{7}{12}(l+k) + \frac{3}{4}p + \frac{3}{8}r - \frac{1}{12}(l'+k').$$

Since by A) $l'+k' \geq p + r/2 - l - k$ we conclude by the above inequality that

$$OPT(L) \leq \frac{2}{3}(l+k+p) + \frac{1}{3}r. \tag{20}$$

By (7),(20) and the inequality $r \geq 2k$ (cf.Figure 1) we finally obtain

$$\frac{4}{3}ISI(L) \geq \frac{2}{3}(l+p) + \frac{4}{9}(k+r) - 4 \geq OPT(L) - 4. \tag{21}$$

B) $p - l' < l + k < p - l' + \frac{r-2k'}{2}$.

If this holds, the best we can hope for is to pack one X-element with one B-item, the first part of the Y-elements by two with an additional B-item and the remaining Y-elements by three. Hence

$$k_{others} \leq p - l' + ((l+k) - (p-l')) + \frac{1}{3}((r-2k') - 2((l+k) - (p-l')))$$

$$= l + k + \frac{1}{3}((r-2k') - 2((l+k) - (p-l')))$$

and by (18) and (8) this yields

$$OPT(L) \leq \frac{2}{3}(l+k+p) + \frac{r}{3}.$$

Clearly this is the same upper bound as discussed in (20) and so (21) also holds.

C) $p - l' - (r-2k') < l + k \leq p - l'$.

If this holds, it follows that $r - 2k' - (p - l' - (l+k)) > 0$ and so the best we can hope for is to pack $l+k$ X-items with one additional B-item, the remaining part of the X-items, i.e. $p - l' - (l+k)$, with one additional Y-item and the rest of the Y-items, i.e. $r - 2k' - (p - l' - (l+k))$, by three. Hence

$$k_{others} \leq l + k + p - l' - (l+k) + \frac{r - 2k' - (p - l' - (l+k))}{3}$$

$$= p - l' + \frac{r - 2k' - (p - l' - (l+k))}{3}$$

and by (18) and (8) this implies

$$OPT(L) \leq \frac{2}{3}(p+l+k) + \frac{r}{3}.$$

Clearly this is the same upper bound as discussed in (20) and so (21) also holds.

D) $l + k \leq p - l' - (r - 2k')$.

If this holds, the best we can hope for is to pack $l + k$ X-items with one additional B-item, $r - 2k'$ X-items with one additional Y-item and the remaining part of the X-items, i.e. $p - l' - (l + k) - (r - 2k')$, by two. Hence

$$k_{others} \leq l + k + r - 2k' + \frac{p - l' - (l + k) - (r - 2k')}{2}$$

and by (18) and (8) this implies

$$OPT(L) \leq \frac{5}{6}(l + k) + \frac{r}{2} + \frac{p}{2} + \frac{1}{6}(l' - 2k'). \tag{22}$$

By D) it follows that $l' - 2k' \leq p - r - l - k$ and substituting this in (22) yields

$$OPT(L) \leq \frac{2}{3}(l + k + p) + \frac{r}{3}.$$

Clearly this is the same upper bound as discussed in (20) and so (21) also holds. This last subcase concludes the proof of our result.

□

In the above lemma we proved that $R_{ISI} \geq 3/4$. Furthermore, this lower bound can be achieved, as will be shown by the next result.

Theorem 8. $R_{ISI} = 3/4$.

Proof. Consider the lists L_n with

$$L_n = \left(\frac{1}{3} + \varepsilon_n, \frac{1}{3} + \varepsilon_n, \underbrace{\frac{1}{3} - 2\varepsilon_n, \frac{1}{3} - 2\varepsilon_n, \ldots, \frac{1}{3} - 2\varepsilon_n}_{12n+1 \text{ times}}, \underbrace{6\varepsilon_n, 6\varepsilon_n, \ldots, 6\varepsilon_n}_{4n \text{ times}} \right).$$

It is easy to verify that

$$OPT(L_n) = 4n + 1.$$

If ε_n is chosen in such a way that the first two items together with the last $4n$ items do not fill the first bin then

$$ISI(L_n) = 3n + 1.$$

Hence $\lim_{n \to \infty} \frac{ISI(L_n)}{OPT(L_n)} = \frac{3}{4}$ and by Lemma 2 the desired result follows.

□

As a last remark we note that for the above lists the ISI-packing is essentially the same as the simple packing.

4 Open question

It would be interesting to find a heuristic with a performance ratio greater than 3/4.

References

[1] Assmann, S.F.: Problems in Discrete Applied Mathematics, Ph.D. Thesis, Mathematics Department, MIT, Cambridge, MA, 1983.

[2] Assmann S. F., Johnson D. S., Kleitman D. J., Leung J. Y.-T.: On a dual version of the one-dimensional bin packing problem, J. of Algorithms 5(1984), 502-525.

[3] Deyermeyer, B.L., Friesen, D.K., Langston, M.A.: Scheduling to maximize the minimum processor finish time in a multiprocessor system, SIAM J. Alg. Disc. Meth. 3(1982), 190-196.

[4] Graham, R.L.: Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17(1969), 263-269.

Acta Cybernetica **14** (1999) 27-36.

On ± 1-representations of integers

János Demetrovics*, Attila Pethő†and Lajos Rónyai*

This paper is dedicated to Professor Ferenc Gécseg on the occasion of his 60th birthday.

1 Introduction

In public key cryptography cryptosystems employing elliptic curves are playing an important role. Such systems are based on the elliptic version of the discrete logarithm problem. Let $\mathbb{K}$ be a finite field, $E = E(\mathbb{K})$ be an elliptic curve over $\mathbb{K}$ and let $P \in E$. If the binary expansion of $n \in \mathbb{N}$ is

$$n = \sum_{i=0}^{l} b_i 2^i, \quad b_i = 0, 1;\ i = 1, \ldots, l;\ b_l = 1,$$

then one can compute $P(n) = nP$ by using the following algorithm:

1. $P(n) \leftarrow P$
2. for $i \leftarrow 1$ to l do
   $\{P(n) \leftarrow 2P(n)$,
   if $b_{l-i} = 1$ then $P(n) \leftarrow P(n) + P.\}$

This algorithm requires l doubling and $\sum_{i=0}^{l-1} b_i$ addition steps. All operations are performed of course on the curve E. The idea is quite old. In a recipe for integer multiplication it appears in the Egyptian Rhind Papyrus dated from about 1650 B.C.

Observing that addition and subtraction on an elliptic curve have the same complexity, Morain and Olivos [MO] developed another algorithm for the computation of $P(n)$. Their algorithm uses one of the representations

$$n = \sum_{i=0}^{l'} d_i 2^i, \quad d_i = -1, 0, 1;\ i = 1, \ldots, l';\ d_{l'} = 1, \tag{1}$$

*Computer and Automation Institute, Hungarian Academy of Sciences, Budapest. The support of OTKA grants 016503, 016526, EC Grant ALTEC-KIT and FKFP grants 0612/1997, 0206/1997 is gratefully acknowledged.

†Institute of Mathematics, Kossuth Lajos University, Debrecen. Research supported in part by the Hungarian Foundation for Scientific Research, Grant N0. 25157/98, and by FKFP grant 0612/1997.

which we shall call a ± 1-representation of n. In the above algorithm only the conditional statement should be changed to

if $d_{l'-i} \neq 0$ then $P(n) \leftarrow P(n) + d_{l'-i}P$.

The new algorithm requires l' doubling and $\sum_{i=0}^{l'-1} |d_i|$ addition/subtraction steps on the curve. As $\sum_{i=0}^{l'-1} |d_i|$ can be considerably smaller than $\sum_{i=0}^{l-1} b_i$, the algorithm of Morain and Olivos may be more efficient, if l' is not too big compared to l.

We will point out to another application of ± 1-representation of integers. Let $A = (a_{ij})_{i=1,2;j=1,2}$ be a matrix with entries from a commutative ring, and with determinant ± 1. As $A^{-1} = \det(A)A^T$ the computation of A^{-1} means in this case only the swapping of $a_{1,2}$ and $a_{2,1}$, and the replacement of the sign of entries of A^T, whenever $\det(A) = -1$.

In contrast to the binary expansion, the ± 1-representation of integers is not at all unique. If, for example, the bitsequence of the binary expansion of n looks like $x01$, then $x0(-1)^k1$ are ± 1-representations of n for all $k \geq 0$. (Here we listed the digits in reverse compared to the usual representation.) Morain and Olivos [MO] (see also Müller [M]) describes two substitutions: $1^k \rightarrow -10^{k-1}1, k \geq 2$ and $1^k01^l \rightarrow -10^{k-1}1-10^{l-1}1 \rightarrow -10^{k-1}-10^l1$, which result usually ± 1-representation of smaller weight than the binary expansion. Moreover, both algorithms are linear in $\log n$. On the other hand, the length of the representation can be at most one longer than the shortest representations. For example the numbers $0^k11, k \geq 0$ have weight 2 and length $k+2$, but the algorithms of Morain and Olivos results $0^k - 101$, which has weight 2, but length $k+3$.

We call a ± 1-representation *optimal*, if $l' + \sum_{i=0}^{l'} |d_i|$ is minimal among the ± 1-representations of n. Note that the quantity $l' + \sum_{i=0}^{l'} |d_i|$ is actually one more than the number of additions/subtractions in E required when using the ± 1-representation (1) for computing nP. The aim of the present paper is to prove the following theorem.

Theorem 1 *There exists an algorithm which computes an optimal ± 1-representation of the integer n in $O(\log |n|)$ additions and comparisons.*

The proof of the theorem is constructive, i.e. we present a linear time algorithm for the computation of an optimal ± 1-representation of integers. Our method is the following: first we associate to the integer n an infinite, bipartite, directed acyclic graph $G(n)$ such that the ± 1-representations of n correspond to suitable directed paths in $G(n)$. Next we establish that to find an optimal ± 1-representation it suffices to consider a subgraph of $G(n)$ having at most $2\log_2 n + 5$ nodes. Our problem is actually equivalent to a single source shortest paths problem in this graph, which can be solved fast using a variant of the well known Dijkstra algorithm [D], [CLR].

2 The construction and elementary properties of $G(n)$

Let $0 \neq n \in \mathbb{N}$ and assume that 2^ν is the highest power of 2, which divides n. For each $k \geq 0$ we consider the solutions x of the congruence

$$x \equiv n \pmod{2^k}, \quad -2^k < x < 2^k. \tag{2}$$

This congruence has one solution, $x = n_k = 0$, if $0 \leq k \leq \nu$, and two solutions, if $k > \nu$. In the latter case we denote the solutions by $n_{k,1}$ and $n_{k,2}$ and order them as follows:

$$0 < |n_{k,2}| \leq 2^{k-1} \leq |n_{k,1}| < 2^k. \tag{3}$$

If $|n_{k,1}| = |n_{k,2}| > 0$ (which may happen only if $k = \nu + 1$), then put $n_{k,1} = 2^\nu$ and $n_{k,2} = -2^\nu$. The set of vertices V of $G(n)$ is

$$V = \{(k, n_k) : 0 \leq k \leq \nu\} \cup \{(k, n_{k,1}), (k, n_{k,2}) : k > \nu\}.$$

To lighten notation we shall refer to vertices (k, n_k) simply as n_k and $(k, n_{k,j})$ as $n_{k,j}$. Thus, in the sequel we will use the notations n_k and $n_{k,j}, j = 1, 2$ in two meanings; either as vertices of $G(n)$ or solutions of (2) satisfying the inequalities (3).

The set of edges E of $G(n)$ is the union of three sets, E_1, E_2, E_3, where

$$\begin{aligned} E_3 &= \{e_{k,1} = e_k = (n_k, n_{k+1}) : k = 0, \ldots \nu - 1\}, \\ E_2 &= \{e_{\nu,1} = (n_\nu, n_{\nu+1,1}), e_{\nu,2} = (n_\nu, n_{\nu+1,2})\}, \\ E_1 &= \{e_{k,j,h} = (n_{k,j}, n_{k+1,h}) : n_{k+1,h} = n_{k,j} + \varepsilon_{k,j,h} 2^k \text{ with} \\ &\quad \varepsilon_{k,j,h} \in \{-1, 0, 1\},\ k > \nu\}. \end{aligned}$$

Here $e = (x, y)$ means that the directed edge e joins vertex x to vertex y.

Let $d^-(x)$, (resp. $d^+(x)$) denote the indegree (the outdegree, resp.) of vertex $x \in V$, i.e. the number of edges having x as their endpoint (starting point, resp.). Now we prove the following simple lemma.

Lemma 1 *We have* $d^-(n_k) = d^-(n_{\nu+1,j}) = 1$, *if* $0 < k < \nu + 1, j = 1, 2$ *and* $d^-(n_{k,j}) = j$, *if* $k > \nu + 1, j = 1, 2$.

Proof: The first assertion is obvious. We consider therefore the second one. Let $k > \nu + 1$. An edge ending at vertex $n_{k,j}$, has, by construction, $n_{k-1,1}$ or $n_{k-1,2}$ as its starting point, and hence belongs to the set E_1. Then there exists an $h \in \{1, 2\}$ and $\varepsilon_{k-1,h,j} \in \{-1, 0, 1\}$ such that

$$n_{k,j} = n_{k-1,h} + \varepsilon_{k-1,h,j} 2^{k-1}.$$

Let first $j = 1$. If $\varepsilon = 0$ or $-sg(n_{k,1})$[1] then

$$|n_{k,1} - \varepsilon 2^{k-1}| \geq |n_{k,1}| \geq 2^{k-1} > |n_{k-1.h}|,\ h = 1, 2$$

[1] We denote by $sg(n)$ the sign of the integer n.

by (3). Hence there can be no edges, which correspond to these values of ε, i.e. $d^-(n_{k,1}) \leq 1$. On the other hand, if $\varepsilon = sg(n_{k,1})$ then for $u = n_{k,1} - sg(n_{k,1})2^{k-1}$ we have

$$|u| = |n_{k,1} - sg(n_{k,1})2^{k-1}| < 2^k - 2^{k-1} = 2^{k-1}.$$

This implies that $u = n_{k-1,1}$ or $u = n_{k-1,2}$ and hence $d^-(n_{k,1}) \geq 1$.

Let now be $j = 2$. If $\varepsilon = 0$ or $sg(n_{k,2})$ then

$$|n_{k,2} - \varepsilon 2^{k-1}| \leq |n_{k,2}| < 2^{k-1}$$

by (3). Hence there is one edge either from $n_{k-1,1}$ or from $n_{k-1,2}$ to $n_{k,2}$. If $\varepsilon = -sg(n_{k,2})$ then, as

$$|n_{k,2} - \varepsilon 2^{k-1}| > 2^{k-1},$$

by (3), there is no edge, which corresponds to this value of ε. Thus $d^-(n_{k,2}) = 2$, as asserted. □

Now we associate weights to the edges of $G(n)$. Let

$$w(e) = \begin{cases} 0, & \text{if } e \in E_3, \\ sg(n_{\nu+1,j}), & \text{if } e = (n_\nu, n_{\nu+1,j}) \in E_2, \\ sg(\varepsilon_{k,j,h}), & \text{if } e = (n_{k,j}, n_{k+1,h}) \in E_1. \end{cases}$$

The following lemma shows that the network $G(n)$ has a quite transparent structure.

Lemma 2 *If $k > \nu$, then there exists for every $\varepsilon \in \{-1, 0, 1\}$ exactly one pair of indices $(j, h), 1 \leq j, h \leq 2$ such that $w(e_{k,j,h}) = \varepsilon$. Moreover, for an edge $e_{k,j,h}$ we have $w(e_{k,j,h}) = 0$ if and only if $h = 2$ and $d^+(n_{k,j}) = 1$.*

Remark 1 *The second assertion of Lemma 2 means that if $k > \nu$ then the subgraph of $G(n)$ spanned by the nodes on the k-th and $k+1$-th levels has one of the following two types:*

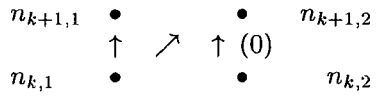


Figure 1

or

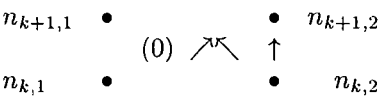


Figure 2

This observation will be important in the computation of an optimal ±1-representation of n.

Proof: We have seen in the proof of Lemma 1 that $w(e_{k,j,2}), j = 1, 2$ takes two different values: 0 and $sg(n_{k+1,2})$, while $w(e_{k,j,1}) = sg(n_{k+1,1})$. But $sg(n_{k+1,2}) \neq sg(n_{k+1,1})$, and both of them are different from 0, implying the first assertion.

¿From what we established so far, we know that $w(e_{k,j,h}) = 0$ implies that $h = 2$. It also follows from Lemma 1 that at level k one of the vertices has outdegree 1, the other has outdegree 2. Having all these, tò prove the second assertion, it suffices to verify that $w(e_{k,j,2}) = 0$ implies that $d^+(n_{k,j}) = 1$. The condition on the edge-weight gives that $n_{k+1,2} = n_{k,j}$. Using this we have

$$|n_{k+1,1} - n_{k,j}| = |n_{k+1,1} - n_{k+1,2}| = 2^{k+1} > \varepsilon 2^k,$$

for any $\varepsilon \in \{1, 0, -1\}$. This means that there can be no edge from $n_{k,j}$ to $n_{k+1,1}$, hence $d^+(n_{k,j}) = 1$. □

We have constructed an, in one direction, infinite directed acyclic graph $G(n)$. Observe, that if $2^k > |n|$, then $n_{k+j,2} = n$ for all $j \geq 1$. We shall prove, that this network describes completely the ±1-representations of the integer n.

To be more precise, let $U(n)$ denote the set of directed paths from 0 to the nodes n_{k,j_k} where $n_{k,j_k} = n$, and let

$$W(n) = \{(w(e_{0,j_1}), \ldots, w(e_{\nu,j_{\nu+1}}), w(e_{\nu+1,j_{\nu+1},j_{\nu+2}}), \ldots, w(e_{k-1,j_{k-1},j_k})),$$
$$\text{where the path } e_{0,j_1}, \ldots, e_{\nu,j_{\nu+1}}, e_{\nu+1,j_{\nu+1},j_{\nu+2}}, \ldots, e_{k-1,j_{k-1},j_k} \in U(n)\}.$$

Remark that $j_i = 1$, if $i \leq \nu$ and $j_i = 1$ or 2, otherwise. Hence $W(n)$ is the set of sequences of weights of directed path from the vertex $n_0 = 0$ to the vertices $n_{k,j_k} = n$.

On the other hand, let

$$E(n) = \{(d_0, \ldots, d_{k-1}), \text{ such that } n = \sum_{i=0}^{k-1} d_i 2^i,\ d_i \in \{-1, 0, 1\}\},$$

i.e. $E(n)$ is the set of sequences of digits of the ±1-representations of n. Now we are in the position to prove the following theorem.

Theorem 2 *If $n \neq 0$, then $W(n) = E(n)$.*

Proof: If $n < 0$ then we have obviously $W(-n) = -W(n)$ and $E(-n) = -E(n)$. Hence it is enough to prove the theorem for $n > 0$, which we assume in the sequel.

Let first

$$s = (w(e_{0,j_1}), \ldots, w(e_{\nu,j_{\nu+1}}), w(e_{\nu+1,j_{\nu+1},j_{\nu+2}}), \ldots, w(e_{k-1,j_{k-1},j_k})) \in W(n).$$

Then

$$\begin{aligned} & e_{0,j_1}, \ldots, e_{\nu,j_{\nu+1}}, e_{\nu+1,j_{\nu+1},j_{\nu+2}}, \ldots, e_{k-1,j_{k-1},j_k} \\ = \ & (n_0, n_1), \ldots, (n_{\nu-1}, n_\nu), (n_\nu, n_{\nu+1,j_{\nu+1}}), \\ & (n_{\nu+1,j_{\nu+1}}, n_{\nu+2,j_{\nu+2}}), \ldots, (n_{k-1,j_{k-1}}, n_{k,j_k}) \in U(n). \end{aligned}$$

We have the relations

$$n_{h,j_h} = n_{h-1,j_{h-1}} + w(e_{h-1,j_{h-1},j_h})2^{h-1},$$

whenever $h > \nu$, by the definition of the vertices and the weights. Hence, as $n_{k,j_k} = n$, we have

$$n = n_{k,j_k} = w(e_{\nu,j_{\nu+1}})2^{\nu} + \sum_{h=\nu+1}^{k-1} w(e_{h,j_h,j_{h+1}})2^h.$$

As $w(e_{h,h+1}) = 0$ for $h = 0, \ldots, \nu - 1$ we obtain

$$n = n_{k,j_k} = \sum_{h=0}^{\nu-1} w(e_{h,h+1})2^h + w(e_{\nu,j_{\nu+1}})2^{\nu} + \sum_{h=\nu+1}^{k-1} w(e_{h,j_h,j_{h+1}})2^h,$$

i.e. $s \in E(n)$.

Assume now that $s = (d_0, \ldots, d_{k-1}) \in E(n)$. Then

$$n = \sum_{i=0}^{k-1} d_i 2^i.$$

Let $n_0 = 0$, and if $0 < h \leq k$, then $n_h = \sum_{i=0}^{h-1} d_i 2^i$. Then $n_k = n$,

$$n_h \equiv n \pmod{2^h}$$

and

$$|n_h| \leq \sum_{i=0}^{h-1} 2^i < 2^h.$$

Thus, if $h > \nu$, then $n_h = n_{h,j_h}$ for $j_h = 1$ or $j_h = 2$. If $h < k$, then

$$n_{h+1} = n_h + d_h 2^h,$$

i.e

$$n_{h+1,j_{h+1}} = n_{h,j_h} + d_h 2^h.$$

This means that there exists an edge from n_{h,j_h} to $n_{h+1,j_{h+1}}$ and its weight is $w(e_{h,j_h,j_{h+1}}) = d_h$. Hence

$$(n_0, n_1), \ldots, (n_{\nu-1}, n_\nu), (n_\nu, n_{\nu+1,j_{\nu+1}}), (n_{\nu+1,j_{\nu+1}}, n_{\nu+2,j_{\nu+2}}), \ldots, (n_{k-1,j_{k-1}}, n_{k,j_k})$$

is a directed path from $n_0 = 0$ to $n_{k,j_k} = n$, i.e. $s \in W(n)$. The proof is complete. □

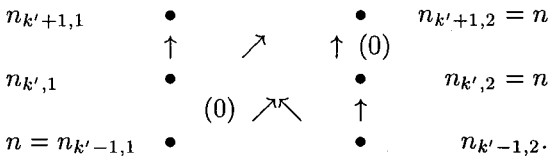
We define the *weight* of a node $n_{k,j}$ of $G(n)$ to be the minimum of the sums $\sum_{j=1}^{k} |w(e^j)|$, where the edges $e^1, e^2, \ldots, e^k$ form a directed path from 0 to $n_{k,j}$. We denote the weight of $n_{k,j}$ by $w(n_{k,j})$. In view of Theorem 2, our task of finding

an optimal representation of n is equivalent to finding a node $n_{k,j}$ with $k + w(n_{k,j})$ minimal among the nodes with $n_{k,j} = n$. (Please note that for an optimal node $n_{k,j}$ a shortest path from 0 to $n_{k,j}$ can not end with an edge of weight zero, hence for the ±1-representation of type (1) obtained from the path we have $l' = k$.)

To solve the latter optimization problem, we examine closely the lowest level of $G(n)$ where n appears as a vertex. Let $k' = \lfloor \log n \rfloor + 2$. Then $2^{k'-2} \leq n < 2^{k'-1}$. Hence $n_{k'-1,1} = n_{h,2} = n$ for all $h \geq k'$, by (3). Moreover, only these vertices are equal to n.

Lemma 3 *If $w(n_{k'-1,1}) \leq w(n_{k'-1,2}) + 1$ then (a shortest path from 0 to) node $n_{k'-1,1}$ provides an optimal representation of n. If $w(n_{k'-1,1}) > w(n_{k'-1,2}) + 1$ then (a shortest path to) node $n_{k',2}$ provides an optimal representation of n.*

Proof: By Remark 1. the layer of $G(n)$ comprising levels $k'-1, k'$ and $k'+1$ looks like



Here the edges without label have weight ±1. Using the fact that a directed path from 0 to a node at level $h \geq k'$ must pass through level $k'-1$, we have

$$w(n_{h,j}) \geq \min\{w(n_{k'-1,1}), w(n_{k'-1,2}) + 1\} = w(n_{k',2}).$$

Hence if $h > k'$ then $h + w(n_{h,j}) > k' + w(n_{k',2})$. ¿From this we see that the optimum is attained at node $n_{k'-1,1}$ or $n_{k,2}$.

If $w(n_{k'-1,1}) \leq w(n_{k'-1,2}) + 1$ then

$$k' - 1 + w(n_{k'-1,1}) \leq k' - 1 + w(n_{k',2}) < k' + w(n_{k',2}),$$

hence $n_{k'-1,1}$ is the (only) optimal node.

On the other hand, if $w(n_{k'-1,1}) > w(n_{k'-1,2}) + 1$ then $w(n_{k'-1,1}) > w(n_{k',2})$, and therefore $k' - 1 + w(n_{k'-1,1}) \geq k' + w(n_{k',2})$. In this case $n_{k',2}$ is an optimal node. □

Note that the lemma implies in particular that the length l' of an optimal ±1-representation (1) of n can have at most two values. The second alternative of the Lemma 3 allows for the possibility of two optimal nodes. This may indeed happen, as exemplified by the representations $7 = 4 + 2 + 1$ and $7 = 8 - 1$.

Proof of Theorem 1. The algorithm now is quite straightforward to outline. On input $n > 0$ we build the the first k' layers of the graph $G(n)$ and calculate the the edge weights. It is a directed acyclic graph (dag) with no more than $2\log_2 n + 5$ vertices and $3\log_2 n + 6$ edges. Following the definition directly, this graph can be built using $O(\log n)$ elementary operations.

By Lemma 3 it suffices to compute the weights $w(n_{k'-1,1})$, $w(n_{k',2})$ together with with an appropriate path from 0 to $n_{k'-1,1}$ or to $n_{k',2}$ which provides the optimal weight. We can use here Dijkstra's algorithm for the single source shortest path problem. In doing this, we have to work with the absolute values of the original edge-weights. Dijkstra's method can be implemented in linear time for dag-s (see for example section 25.4 in [CLR]), hence this phase can also be accomplished in time $O(\log n)$. □

3 A detailed algorithm

Here we present a detailed and streamlined procedure which performs the tasks sketched in the proof of Theorem 1. It computes an optimal ± 1-representation of the input integer $n > 0$. In the following description we use a two-dimensional array $n(h,j), j = 1,2$, to represent the vertices $n_{h,j}$ of the network $G(n)$. The value of $n(h,j)$ is a three-dimensional vector, whose i-th coordinate will be denoted by $n(h,j)[i]$.

Upon termination $n(h,j)[3]$ will store $w(n_{h,j})$. Moreover, $n(h,j)[1]$ stores an identifier of the next to last vertex of an optimal path to $n_{h,j}$, and $n(h,j)[2]$ contains the weight of the last edge along this path. More formally, in the general situation (i.e. if $h > \nu$) we intend to achieve the following:

$$
\begin{aligned}
n(h,j)[3] &= \min\{n(h-1,\ell)[3] + |w(e_{h-1,j,\ell})|, \text{ where } e_{h-1,j,\ell} \in G(n)\},\\
n(h,j)[2] &= w(e_{h-1,j,\ell}), \text{ if } n(h,j)[3] = n(h-1,\ell)[3] + |w(e_{h-1,j,\ell})|,\\
n(h,j)[1] &= \ell, \text{ if } n(h,j)[3] = n(h-1,\ell)[3] + |w(e_{h-1,j,\ell})|.
\end{aligned}
$$

Algorithm

Input: $n > 0$ an integer
Output: $(d_0, \ldots, d_{k-1})$ an optimal ± 1-representation of n.

1. $k' := \lfloor \log n \rfloor + 2$
2. Compute $G(n)$ up to level k'
3. **for** $h := 1$ **to** ν **do** $n(h,1) := (1,0,0)$
4. $n(\nu+1,1) := (1, w(e_{\nu,\nu+1,1}), 1); n(\nu+1,2) := (1, w(e_{\nu,\nu+1,2}), 1)$
5. **for** $h := \nu + 2$ **to** k' **do**
   if $e_{h-1,1,1} \in G(n)$ **then begin**
   $n(h,1) := (1, w(e_{h-1,1,1}), n(h-1,1)[3] + 1)$
   $n(h,2) := (2, 0, n(h-1,2)[3])$
   if $n(h-1,1)[3] + 1 < n(h,2)[3]$ **then**
   $n(h,2) := (1, w(e_{h-1,1,2}), n(h,1)[3])$
   end

else begin
$n(h,1) := (2, w(e_{h-1,2,1}), n(h-1,2)[3]+1)$
$n(h,2) := (1, 0, n(h-1,1)[3])$
if $n(h-1,2)[3]+1 < n(h,2)[3]$ **then**
$n(h,2) := (2, w(e_{h-1,2,2}), n(h,1)[3])$
end

6. $k := k'-1; j := n(k,1)[1]; d := (n(k,1)[2])$
if $n(k',2)[3] < n(k,1)[3]$ **then** $k := k'; j := n(k,2)[1]; d := (n(k,2)[2])$

7. **while** $k \neq 0$ **do**
d:=(n(k,j)[2],d); j:=n(k,j)[1]; k:=k-1

8. **output** d.

Proposition 1 *The preceding Algorithm computes an optimal ±1-representation of the integer n in $O(\log n)$ time.*

Proof: It is clear that the algorithm terminates after $O(\log n)$ steps. Therefore, it is enough to establish correctness.

The basis of the calculation of $w(n_{h,j})$ is the straightforward relation

$$w(n_{h,j}) = \min\{w(n_{h-1,j}) + |w(e_{h-1,i,j})|, \text{ where } \quad (n_{h-1,i}, n_{h,j}) \in G(n)\}.$$

As $w(e) = 0$ for $e \in E_3$, we have $w(n_{h,j}) = 0$ for $h \leq \nu$. Thus $n(h,1)[3]$ is set correctly in Step 3 for $1 \leq h \leq \nu$. The same is true for $n(\nu+1, j)[3], j = 1, 2$, because $|w(e)| = 1$ for $e \in E_2$. If $h \geq \nu + 2$ then the h-th level of $G(n)$ has one of the shapes, presented on Figures 1 and 2. The value of $n(h,j)[3]$ is determined in Step 5 according to these alternatives. Thus $n(h,j)[3] = w(n_{h,j})$ for all h and j considered. By Lemma 3 it is enough to compute the weights up until level $\lfloor \log n \rfloor + 2$, hence k' is set properly in Step 1.

Lemma 3 shows also that in Step 6 the parameters k, j of an optimal node $n_{k,j}$ are calculated correctly. In fact, we set $k = k' - 1$, if $w(n_{k'-1,1}) \leq w(n_{k',2})$, and $k = k'$, if $w(n_{k'-1,1}) > w(n_{k',2})$. Finally, by tracing backwards an optimal path to $n_{k,j}$ in loop 7, we compute the digits of an optimal ±1-representation. The proposition is proved. □

Acknowledgement We thank I. Ruzsa and S. Turjányi for their comments and helpful conversations during the preparation of this paper.

References

[CLR] T.H. CORMEN, C.E.LEISERSON and R.L. RIVEST, *Introduction to algorithms*, The MIT Press, 1990.

[D] E.W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math. **1** (1959), 269-271.

[Me] A. MENEZES, *Elliptic curve public key cryptosystems*, Kluwer Academic Publishers, 1993.

[M] V. MÜLLER, *Efficient algorithms for multiplication on elliptic curves* to appear.

[MO] F. MORAIN and J. OLIVOS, *Speeding up the computations on an elliptic curve using addition-subtraction chains*, in F. MORAIN, *Courbes elliptiques et tests de primalité*, Doctoral Thesis, Université Lyon I, 1990.

Acta Cybernetica **14** (1999) 37-50.

Complete Finite Automata Network Graphs with Minimal Number of Edges*

Pál Dömösi† Chrystopher L. Nehaniv‡

Dedicated to Professor Ferenc Gécseg on his 60th birthday

Abstract

An automata network graph is said to be n-complete (under projection) if every automata network having underlying graph with n vertices can be simulated (under projection) on it. In this paper n-complete automata network graphs with minimal number of edges are completely characterized.

1 Basic Notions

Let $f : X_1 \times \ldots \times X_n \to X$ be a mapping having n variables for some positive integer n, moreover, let $t \in \{1, \ldots, n\}$. f is said to *really depend* on its t^{th} variable if there exist $x_1 \in X_1, \ldots, x_{t-1} \in X_{t-1}, x_t, x_t' \in X_t, x_{t+1} \in X_{t+1}, \ldots, x_n \in X_n$ having $f(x_1, \ldots, x_n) \neq f(x_1, \ldots, x_{t-1}, x_t', x_{t+1}, \ldots, x_n)$. If f does not have this property then we also say that f is *really independent* of its t^{th} variable. Moreover, if there is no danger of confusion then sometimes we omit the attribute "really".

For a given non-empty set X and positive integer n denote by X^n the n^{th} 0 power of X. Given a k-element subset H of $\{1, \ldots, n\}$, $H = \{i_1, \ldots, i_k\}$ $(i_1 < \ldots < i_k)$, the *H-projection* of X^n is a mapping $pr_H : X^n \to X^k$ defined by $pr_H(x_1, \ldots, x_n) = (x_{i_1}, \ldots, x_{i_k})$, where $(x_1, \ldots, x_n) \in X^n$. The function $pr_H(F) : X^k \to X^k$ with $pr_H(F(x_1, \ldots, x_n)) = pr_H(F)(pr_H(x_1, \ldots, x_n)), (x_1, \ldots, x_n) \in X^n$ is called the *H-projection* of $F : X^n \to X^n$ (if it exists). If $H = \{h\}$ for

*This work was supported by grants of the University of Aizu "Algebra & Computation" and "Automata Networks" projects (R-10-1, R-10-4), the "Automata & Formal Languages" project of the Hungarian Academy of Sciences and Japanese Society for Promotion of Science (No. 15), the Academy of Finland (No 137358), the Hungarian National Foundation for Scientific Research (OTKA T019392), and the Higher Education Research Foundation of the Hungarian Ministry of Education (No. 222).

†L. Kossuth University, Institute of Mathematics and Informatics, 4032 Debrecen, Egyetem tér 1, Hungary, e-mail: domosi@@math.klte.hu

‡School of Computer Science & Engineering University o f Aizu, Aizu-Wakamatsu City 965, Japan, e-mail: nehaniv@@u-aizu.ac.jp

some $h \in \{1,\ldots,n\}$, i.e., H is a singleton then sometimes we use the expression *h-projection* (of a vector or function) in the same sense as the concept "H-projection". (And in this case sometimes we use the notation pr_h instead of $pr_{\{h\}}$.) Moreover, for an arbitrary $i \in \{1,\ldots,n\}$, we define the i^{th} *component* of $F : X^n \to X^n$ as the function $cp_i(F) : X^n \to X$ with $cp_i(F)((x_1,\ldots,x_n)) = pr_i(F(x_1,\ldots,x_n))$ $(x_1,\ldots,x_n) \in X^n$.

For any pair $F_i : X^n \to X^n, i = 1,2$, one denotes by $F_1 \circ F_2 : X^n \to X^n$ the function $F_1 \circ F_2(x_1,\ldots,x_n) = F_1(F_2(x_1,\ldots,x_n))$, $(x_1,\ldots,x_n) \in X^n$.

A (finite) *directed graph* (or, in short, a *digraph*) $\mathcal{D} = (V,E)$ (of order $n > 0$) is a pair of the sets of *vertices* $V = \{v_1,\ldots,v_n\}$ and *edges* $E \subseteq V \times V$. $v_i \in V$ is an *isolated vertex* if $(\{v_i\} \times V \cup V \times \{v_i\}) \cap E = \emptyset$. If $(v_i,v_j) \in E$ and $i = j$ then (v_i,v_j) is called *(self-) loop edge*. The digraph $\mathcal{D}' = (V',E')$ is a *subdigraph* of $\mathcal{D}$ if V' is a non-void subset of V, and $E' \subseteq E$. $\mathcal{D}$ is said to be *connected* for $v_i \in V$ if every vertex $v_j \in V$ has a (directed) path from v_i to v_j. $\mathcal{D}$ is called *strongly connected* if it is connected for all of its vertices. Moreover, $\mathcal{D}$ is *centralized* if there exists a $v_i \in V$ with $V \times \{v_i\} \subseteq E$ (including $(v_i,v_i) \in E$). In addition, a digraph $\mathcal{D} = (V,E)$ having a structure $V = \{v_1,\ldots,v_n\}$, $E = \{(v_i,v_{i+1(modn)}) : i = 1,\ldots,n\}$ is called a *cycle (with n length)*. We also say that a digraph $\mathcal{D}$ *has a cycle (with n length)* if there is a subdigraph of $\mathcal{D}$ which forms a cycle (with n length). A transformation $F : X^n \to X^n$ is said to be *compatible* with a digraph $\mathcal{D} = (V,E)$ (of order n) if F has the form $F(x_1,\ldots,x_n) = (f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n))$ $((x_1,\ldots,x_n) \in X^n)$ and $f_i : X^n \to X$, $i = 1,\ldots,n$ may depend only on x_i and those x_j for which $(v_j,v_i) \in E$ (including the case $i = j$).

A *word* (over X) is a finite sequence of elements of some finite non-empty set X. We call the set X an *alphabet,* the elements of X *letters*. If u and v are words over an alphabet X, then their *catenation* uv is also a word over X. Especially, for every word u over X, $u\lambda = \lambda u = u$, where λ denotes the empty word having no letters. The *length* $|w|$ of a word w is the number of letters in w, where each letter is counted as many times as it occurs. Thus $|\lambda| = 0$. By the *free monoid* X^* *generated by* X we mean the set of all words (including the *empty word* λ) having catenation as multiplication. We set $X^+ = X^* \setminus \{\lambda\}$, where the subsemigroup X^+ of X^* is said to be *free semigroup generated by* X.

By an *automaton* $\mathcal{A} = (A,X,\delta)$ we mean a finite automaton without outputs. Here A is the (finite non-empty) *state set*, X is the *input alphabet* and $\delta : A \times X \to A$ is the *transition function*. We also use δ in an extended sense, i.e., as a mapping $\delta : A \times X^* \to A$, where $\delta(a,\lambda) = a$ $(a \in A)$ and $\delta(a,px) = \delta(\delta(a,p),x)$ $(a \in A, p \in X^*, x \in X)$. For a given word $p \in X^*$, the *transition induced by* p is the function $\delta_p : A \to A$ that takes any state $a \in A$ to $\delta(a,p)$.

If $A = Z^n$ for some $|Z| \geq 1$ and $n \geq 1$ (where $|Z|$ denotes the cardinality, i.e., the number of elements in Z) then we say that $\mathcal{A}$ is a *finite state-0 automata network (of size n with respect to the basic local state set Z)*. Then the *underlying graph* $\mathcal{D}_\mathcal{A} = (V_\mathcal{A},E_\mathcal{A})$ of $\mathcal{A}$ is defined by $V_\mathcal{A} = \{1,\ldots,n\}, E_\mathcal{A} = \{(i,j) \mid \exists x \in X : cp_j(\delta_x)$ really depends on its i^{th} variable$\}$. $\mathcal{A}$ is a $\mathcal{D}$-*network* if $\mathcal{D} = (V,E)$ is a digraph with $V = V_\mathcal{A}$ and $E \supseteq E_\mathcal{A}$. In other words, $\mathcal{A}$ is a $\mathcal{D}$-network if every mapping $\delta_x : A \to A$ $(x \in X)$ is *compatible* with $\mathcal{D}$. Note that a size n automata network

may be regarded as comprising n component automata $\mathcal{A}_i = (Z, Z^n \times X, \delta_i)$, $i \in \{1, \ldots, n\}$, where the δ_i are defined by

$$\delta(z, x) = (\delta_1(z_1, (z, x)), \ldots, \delta_n(z_n, (z, x))),$$

for $z = (z_1, \ldots, z_n) \in Z^n$, $x \in X$. One may of course suppress the components of Z^n in the inputs to $\mathcal{A}_i$ upon which δ_i does not really depend.

If $n = 1$ or $|Z| = 1$ then we say that $\mathcal{A} = (Z^n, X, \delta)$ is a *trivial* automata network. The purpose of this paper is to investigate the state-homogeneous automata networks having state sets of the form Z^n, for a positive integer $n > 1$ and fixed finite set Z of cardinality at least two. Therefore, by an automata network we shall mean a non-trivial finite state-homogeneous network.

Let $\mathcal{A} = (Z^n, X, \delta), \mathcal{B} = (Z^m, Y, \delta')$ be networks (having the same basic set Z). We say that $\mathcal{B}$ *simulates* $\mathcal{A}$ *by projection* if there exists an $H \subseteq \{1, \ldots, m\}$ such that every $\delta_x : Z^n \to Z^n$ $(x \in X)$ is an H-projection of a mapping $\delta'_p : Z^m \to Z^m$ $(p \in Y^+)$. If there exists a $\mathcal{D}$-network $\mathcal{B}$ which simulates a given network $\mathcal{A}$ by projection then it is said that $\mathcal{A}$ *can be simulated* on $\mathcal{D}$ *by projection.* A digraph $\mathcal{D}$ is called *n-complete (with respect to simulation by projection)* if every network of size n can be simulated on $\mathcal{D}$ by projection. The n-complete digraph $\mathcal{D} = (V, E)$ has *minimal number of edges* if for every n-complete digraph $\mathcal{D}' = (V', E')$, $|V| = |V'|$ implies $|E| \leq |E'|$.

2 Preliminary results

We start with the following technical result.

Lemma 2.1.(see [2]) *Given a finite group G, a positive integer $n > 1$, let us define for every distinct $i, j \in \{1, \ldots, n\}$ the functions $F^{(t)}_{i,j} : G^n \to G^n, t = 1, 2, 3$, $F^{(4)}_j : G^n \to G^n$, and $U_{i,j} : G^n \to G^n$ as follows.*

$$F^{(1)}_{i,j}(g_1, \ldots, g_n) = (g_1, \ldots, g_{j-1}, g_i g_j, g_{j+1}, \ldots, g_n),$$

$$F^{(2)}_{i,j}(g_1, \ldots, g_n) = (g_1, \ldots, g_{j-1}, {g_i}^{-1} g_j, g_{j+1}, \ldots, g_n),$$

$$F^{(3)}_{i,j}(g_1, \ldots, g_n) = (g_1, \ldots, g_{j-1}, g_i, g_{j+1}, \ldots, g_n),$$

$$F^{(4)}_j(g_1, \ldots, g_n) = (g_1, \ldots, g_{j-1}, {g_j}^{-1}, g_{j+1}, \ldots, g_n),$$

$$U_{i,j}(g_1, \ldots, g_n) = (g_1, \ldots, g_{i-1}, g_j, g_{i+1}, \ldots, g_{j-1}, g_i, g_{j+1}, \ldots, g_n).$$

Then for arbitrary, pairwise distinct $i, j, k \in \{1, \ldots, n\}$ we get

$$F^{(1)}_{i,j} = F^{(2)}_{i,k} \circ F^{(1)}_{k,j} \circ F^{(1)}_{i,k} \circ F^{(2)}_{k,j},$$

$$F_{i,j}^{(2)} = F_{i,k}^{(1)} \circ F_{k,j}^{(1)} \circ F_{i,k}^{(2)} \circ F_{k,j}^{(2)},$$
$$F_{i,j}^{(3)} = F_{k,j}^{(1)} \circ F_{i,k}^{(1)} \circ F_{k,j}^{(2)} \circ F_{i,k}^{(2)} \circ F_{k,j}^{(2)} \circ F_{k,j}^{(3)},$$
$$U_{i,j} = F_j^{(4)} \circ F_{i,j}^{(1)} \circ F_j^{(4)} \circ F_i^{(4)} \circ F_{j,i}^{(2)} \circ F_{i,j}^{(1)}.$$

□

Given a non-void set Y, a positive integer n, let $\mathcal{T}_Y$ denote the full transformation semigroup of all functions from Y to Y. In addition, for every subset $H \subseteq \mathcal{T}_Y$, let $< H >$ denote the subsemigroup of $\mathcal{T}_Y$ generated by H. Moreover, for any finite set X with $|X| > 1$ and positive integer $n > 1$, denote $\mathcal{T}_{X,n}$ the subsemigroup of all transformations of $\mathcal{T}_{X^n}$ having the form $F(x_1, \ldots, x_n) = (x_{t(1)}, \ldots x_{t(n)}), (x_1, \ldots, x_n) \in X^n, t : \{1, \ldots, n\} \to \{1, \ldots, n\}$, and let

$$\Gamma_{X^n} = \{F : X^n \to X^n \mid F(x_1, \ldots, x_n) = (x_1, \ldots, x_{i-1}, f(x_i, x_j), x_{i+1}, \ldots, x_n), \text{ where } f : X^2 \to X, i, j \in \{1, \ldots, n\}, (x_1, \ldots, x_n) \in X^n\},$$

(It is understood that the case $i = j$ is allowed in the above definition of Γ_{X^n}.) Define the *elementary collapsing* $t_{j,k} : \{1, \ldots, n\} \to \{1, \ldots, n\}$ for $1 \leq j \neq k \leq n$,

$$t_{j,k}(i) = \begin{cases} j & \text{if } i = k \\ i & \text{otherwise} \end{cases}.$$

Moreover, as usual we say that $u_{j,k} : \{1, \ldots, n\} \to \{1, \ldots, n\}$ for $1 \leq j \neq k \leq n$ is a *transposition* if

$$u_{j,k}(i) = \begin{cases} j & \text{if } i = k \\ k & \text{if } i = j \\ i & \text{otherwise} \end{cases}.$$

Let $\mathcal{F}_{X^{n-1} \times \{d\}}$ be the semigroup of functions $\{F \in \mathcal{T}_{X^n} : F(x_1, \ldots, x_n) \in X^{n-1} \times \{d\}, x_1, \ldots, x_n \in X, F$ is really independent of its last variable$\}$.

Lemma 2.2. (see [2]) $\mathcal{F}_{X^{n-1} \times \{d\}} \subsetneq < \Gamma_{X^n} >$.

Proof. Fix arbitrary $c \neq d \in X$ and let $(c_1, \ldots, c_{n-1}) \in X^{n-1}$,

$$F_{(c_1,\ldots,c_{n-1})}(x_1, \ldots, x_n) = \begin{cases} (x_1, \ldots, x_n) & \text{if } (x_1, \ldots, x_n) = (c_1, \ldots, c_{n-1}, c), \\ (x_1, \ldots, x_{n-1}, d) & \text{if } (x_1, \ldots, x_n) \neq (c_1, \ldots, c_{n-1}, c) \end{cases}$$

$((x_1, \ldots, x_n) \in X^n)$. First we prove that $F_{(c_1,\ldots,c_{n-1})} \in < \Gamma_{X^n} >$.

If $n = 2$, then our statement holds by definition. Otherwise, $n > 2$ and for every $b \in X$, define

$$F_b^{(0)}(x_1, \ldots, x_n) = \begin{cases} (x_1, \ldots, x_{n-1}, c) & \text{if } x_{n-1} = b, x_n = c, \\ (x_1, \ldots, x_{n-1}, d) & \text{otherwise,} \end{cases}$$

where $(x_1, \ldots, x_n) \in X^n$.

For every $i \in \{1, \ldots, n-1\}, (c_i, \ldots, c_{n-1}) \in X^{n-i}$, let

$$F_{(c_i,\ldots,c_{n-1})}(x_1, \ldots, x_n) = \begin{cases} (x_1, \ldots, x_{n-1}, c) & \text{if } (x_i, \ldots, x_n) = (c_i, \ldots, c_{n-1}, c), \\ (x_1, \ldots, x_{n-1}, d) & \text{otherwise,} \end{cases}$$

where $x = (x_1, \ldots, x_n) \in X^n$). It is clear that $F_{(c_{n-1})} = F^{(0)}_{c_{n-1}}$. On the other hand, for every $i \in \{2, \ldots, n-1\}$, $F_{(c_{i-1},\ldots,c_{n-1})} = F_{(c_i,\ldots,c_{n-1})} \circ U_{i-1,n-1} \circ F^{(0)}_{c_{i-1}} \circ U_{i-1,n-1}$. Simultaneously, we have by definition that $F^{(0)}_{c_{i-1}} \in \Gamma_{X^n}$ holds for every $i \in \{2, \ldots, n-1\}$. Moreover, using Lemma 2.1, it can be shown easily $U_{i,j} \in < \Gamma_{X^n} >$. Thus we get our statement by induction.

Now we consider a pair $(c_1, \ldots, c_{n-1}), (d_1, \ldots, d_{n-1}) \in X^{n-1}, d \in X$, ($(c_1, \ldots, c_{n-1}) = (d_1, \ldots, d_{n-1})$ is allowed), and define

$$F^{(1)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}(x) = \begin{cases} (c_1, \ldots, c_{n-1}, d) & \text{if } (x_1, \ldots, x_{n-1}) = (d_1, \ldots, d_{n-1}), \\ (d_1, \ldots, d_{n-1}, d) & \text{if } (x_1, \ldots, x_{n-1}) = (c_1, \ldots, c_{n-1}), \\ (x_1, \ldots, x_{n-1}, d) & \text{otherwise,} \end{cases}$$

$$F^{(2)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}(x) = \begin{cases} (c_1, \ldots, c_{n-1}, d) & \text{if } (x_1, \ldots, x_{n-1}) = (d_1, \ldots, d_{n-1}), \\ (x_1, \ldots, x_{n-1}, d) & \text{otherwise,} \end{cases}$$

where $x = (x_1, \ldots, x_n) \in X^n$.

Next we show $F^{(i)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} \in < \Gamma_{X^n} >, i = 1, 2$.

We have $c \in X$ arbitrary with $c \neq d$ and set $F^{(3)}_c(x) = (x_1, \ldots, x_{n-1}, c)$, $F^{(3)}_d(x) = (x_1, \ldots, x_{n-1}, d)$, and

$$F^{(4)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}(x) = \begin{cases} (c_1, \ldots, c_{n-1}, c) & \text{if } (x_1, \ldots, x_n) = (d_1, \ldots, d_{n-1}, c), \\ (x_1, \ldots, x_{n-1}, d) & \text{otherwise} \end{cases}$$

where $x = (x_1, \ldots, x_n) \in X^n, c, d \in X, c \neq d$, moreover, consider $F_{(c_1,\ldots,c_{n-1})}$ as before. In addition, let

$$F^{(5)}(x_1, \ldots, x_n) = \begin{cases} (x_1, \ldots, x_{n-1}, c) & \text{if } x_n = d, \\ (x_1, \ldots, x_{n-1}, d) & \text{if } x_n = c, \\ (x_1, \ldots, x_{n-1}, x_n) & \text{otherwise,} \end{cases}$$

and let for every $a \in X$,

$$F^{(6)}_a(x_1, \ldots, x_n) = \begin{cases} (x_1, \ldots, x_{n-2}, a, x_n) & \text{if } x_n = c, \\ (x_1, \ldots, x_{n-1}, x_n) & \text{otherwise} \end{cases}$$

($(x_1, \ldots, x_n) \in X^n$). It is clear that $F^{(3)}_c, F^{(3)}_d, F^{(5)}, F^{(6)}_a \in \Gamma_{X^n}$. Next we show that $F^{(4)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} \in < \Gamma_{X^n} >$. Indeed, by an easy computation we get

$F^{(4)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} = U_{n-2,n-1} \circ \ldots \circ U_{2,n-1} \circ U_{1,n-1} \circ F^{(6)}_{c_1} \circ U_{1,n-1} \circ F^{(6)}_{c_2} \circ U_{2,n-1} \circ \ldots \circ U_{n-3,n-1} \circ F^{(6)}_{c_{n-2}} \circ U_{n-2,n-1} \circ F^{(6)}_{c_{n-1}} \circ F_{(d_1,\ldots,d_{n-1})}$. On the other hand, by Lemma 2.1 we can see easily $U_{i,j} \in < \Gamma_{X^n} >$. But then $F^{(2)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} = F^{(3)}_d \circ F^{(4)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} \circ F^{(3)}_c$ implies $F^{(2)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} \in < \Gamma_{X^n} >$. It remains to prove that $F^{(1)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} \in < \Gamma_{X^n} >$. This connection, completing the proof, comes from $F^{(1)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} = F^{(3)}_d \circ F^{(4)}_{(d_1,\ldots,d_{n-1}),(c_1,\ldots,c_{n-1})} \circ F^{(5)} \circ F^{(4)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})} \circ F^{(3)}_c$.

Finally, for every pair $(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1}) \in X^{n-1}$, let us consider the mappings $F^{(1)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}$, $F^{(2)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}$ defined before. Observe that $F^{(1)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}$ acts as a transposition in the permutation group over the set $X^{n-1} \times \{d\}$, while $F^{(2)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}$ acts as an elementary collapsing in the transformation semigroup over the set $X^{n-1} \times \{d\}$. We have already proved that all of these transpositions and elementary collapsings are in $< \Gamma_{X^n} >$. Moreover, it is well-known that the set of all transpositions and elementary collapsings on a set generates all mappings on that set, so any map taking $X^{n-1} \times \{d\}$ to itself may be written as the restriction to $X^{n-1} \times \{d\}$ of a composite of the the above functions. A moment's reflections shows that the set of all these $F^{(1)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}$, $F^{(2)}_{(c_1,\ldots,c_{n-1}),(d_1,\ldots,d_{n-1})}$ in fact generates all of $\mathcal{F}_{X^{n-1}\times\{d\}}$, since a function in the latter is uniquely determined by its 0 on $X^{n-1} \times \{d\}$. In addition, it is clear that $\Gamma_{X^n} \setminus \mathcal{F}_{X^{n-1}\times\{d\}}$ is non-void. This completes the proof. □

Next we show

Lemma 2.3. *Given a finite group G, a pair of relatively prime integers m, n with $1 \leq m < n$, let us define for every $\ell \in \{1,\ldots,n\}$, the transformations $T^{(0)}_i : G^n \to G^n$, $T^{(k)}_i : G^n \to G^n, k = 1,2,3,4$ as follows.*

$$T^{(0)}(g_1,\ldots,g_n) = (g_n, g_1, \ldots, g_{n-1}),$$

$$T^{(1)}_\ell(g_1,\ldots,g_n) = (g_n, g_1, \ldots, g_{\ell-2}, g_{\ell-m-1(\bmod n)} g_{\ell-1}, g_\ell, \ldots, g_{n-1}),$$

$$T^{(2)}_\ell(g_1,\ldots,g_n) = (g_n, g_1, \ldots, g_{\ell-2}, g^{-1}_{\ell-m-1(\bmod n)} g_{\ell-1}, g_\ell, \ldots, g_{n-1}),$$

$$T^{(3)}_\ell(g_1,\ldots,g_n) = (g_n, g_1, \ldots, g_{\ell-2}, g_{\ell-m-1(\bmod n)}, g_\ell, \ldots, g_{n-1}),$$

$$T^{(4)}_\ell(g_1,\ldots,g_n) = (g_n, g_1, \ldots, g_{\ell-2}, g^{-1}_{\ell-1}, g_\ell, \ldots, g_{n-1}).$$

Then for any fixed $\ell \in \{1,\ldots,n\}$, $\mathcal{T}_{G,n} \subsetneq < \{T^{(0)}, T_\ell^{(k)} : k = 1,2,3,4\} >$.

Proof. For every $i \in \{1,\ldots,n\}$, $k \in \{1,\ldots,4\}$, $T_i^{(k)} = (T^{(0)})^{n+i-\ell} \circ T_\ell^{(k)} \circ (T^{(0)})^{n+\ell-i}$. Thus we shall show only $\mathcal{T}_{G,n} \subsetneq < \{T^{(0)}, T_\ell^{(k)} : \ell \in \{1,\ldots,n\}, k = 1,2,3,4\} >$. It is clear that using the notions in Lemma 2.1, by the simple fact that every permutation is a composite of transpositions, and moreover, transformations can be generated by permutations and elementary collapsings, we obtain $\mathcal{T}_{G,n} \subseteq < \{F_{i,j}^{(3)}, U_{i,j} : i,j \in \{1,\ldots,n\} >$. On the other hand, $< \{T^{(0)}, T_\ell^{(k)} : k = 1,2,3,4, \ell = 1,\ldots,n\} > \backslash \mathcal{T}_{G,n} \neq \emptyset$ is clear. Thus, it is enough to prove that for every $i,j \in \{1,\ldots,n\}$, $F_{i,j}^{(3)}, U_{i,j} \in < \{T^{(0)}, T_\ell^{(k)} : k = 1,2,3,4, \ell = 1,\ldots,n\} >$. Using $F^{(d)}_{i+(j-1)m-1(\bmod n),i+jm-1(\bmod n)} = (T^{(0)})^{n-1} \circ T^{(d)}_{i+jm(\bmod n)}$, $d = 1,2,3, i \in \{1,\ldots,n\}, j = 0,1,\ldots$, by an inductive application of Lemma 2.1, we have $F^{(d)}_{i-m-1(\bmod n),i+jm-1(\bmod n)} \in < \{T^{(0)}, T_\ell^{(k)} : k = 1,2,3,4, \ell = 1,\ldots,n\} >$ $(i \in \{1,\ldots,n,\}, j = 0,1\ldots)$.

Therefore, because m and n are relatively prime, we receive $F_{i,j}^{(d)} \in < \{T^{(0)}, T_\ell^{(k)} : k = 1,2,3,4, \ell = 1,\ldots,n\} >$ $(d = 1,2,3, i,j \in \{1,\ldots,n\})$.

Moreover, we also have $F_{i-1}^{(4)} = (T^{(0)})^{n-1} \circ T_i^{(4)}$, $i \in \{1,\ldots,n\}$. Hence, applying Lemma 2.1 again, we obtain $U_{i,j} \in < \{T^{(0)}, T_\ell^{(k)} : k = 1,\ldots,4, \ell = 1,\ldots,n\} >$, $i,j \in \{1,\ldots,n\}$ and thus, having $F_{i,j}^{(3)} \subseteq < \{T^{(0)}, T_\ell^{(k)} : k = 1,2,3,4\} >$ $(i,j \in \{1,\ldots,n\})$, the proof is complete. □

We shall use the following

Lemma 2.4. (see [1], [2]) *Given a positive integer* n, *let* $G = < g >$ *denote a finite non-trivial cyclic group with a generator* $g \in G$. *There exists an arrangement* $a_1,\ldots,a_m$ $(m = |G|^n)$ *of the elements in the* n^{th} *direct power* G^n *of* G *such that for every* $i = 1,\ldots,m-1$ *there is a* $j \in \{1,\ldots,n\}$ *with* $a_{i+1} \in \{(g_1,\ldots,g_{j-1},g_jg^{-1},g_{j+1},\ldots,g_n),(g_1,\ldots,g_{j-1},g_jg,\ g_{j+1},\ \ldots,g_n)\}$, *whenever* $a_i = (g_1,\ldots,g_n)$ $(\in G^n)$. □

Now we are ready to prove the following key lemma.

Lemma 2.5. *For any fixed* $\ell \in \{1,\ldots,n\}$, $\mathcal{T}_{X^n}$ *is generated by the union of* $< \{T^{(0)}, T_\ell^{(k)} : k = 1,\ldots,4\} >$ *and the set of all functions* $F : X^n \to X^n$ *having the form* $F(x_1,\ldots,x_n) = (x_1,\ldots,x_{\ell-1},f(x_1,\ldots,x_n),x_{\ell+1},\ldots,x_n)$, $f : X^n \to X$, *where* $x_1,\ldots,x_n \in X$.

Proof. We can take out of consideration the trivial case $|X| = 1$. Thus we assume $|X| > 1$.

It is clear that without loss of generality we may suppose $\ell = 1$. On the other hand, using Lemma 2.3, $\{U_{i,j} : i,j \in \{1,\ldots,n\}\} \subsetneq < \{T^{(0)}, T_\ell^{(k)} : k = 1,\ldots,4 >$. Thus it is enough to prove that the union of $\{U_{i,j} : i,j \in \{1,\ldots,n\}\}$ and the set of

all functions $F : X^n \to X^n$ having the form $F(x_1, \ldots, x_n) = (f(x_1, \ldots, x_n), x_2, \ldots, x_n)$, generates $\mathcal{T}_{X^n}$.

For every pair $i \in \{1, \ldots, n\}$, $f : X^n \to X$, define the function $F_{i,f} : X^n \to X^n$ with $F_{i,f}(x_1, \ldots, x_n) = (x_1, \ldots, x_{i-1}, f(x_1, \ldots, x_n), x_{i+1}, \ldots, x_n)$ $(x_1, \ldots, x_n \in X)$. Thus, by letting $f' = f \circ U_{i,j}$, we have $F_{j,f} = U_{i,j} \circ F_{i,f'} \circ U_{i,j}$. So for every pair $i \in \{1, \ldots, n\}$, $f : X^n \to X$, $F_{i,f} \in < \mathcal{T}_{X,n} \cup \{F : X^n \to X^n \mid F(x_1, \ldots, x_n) = (f(x_1, \ldots, x_n), x_2, x_3, \ldots, x_n), f : X^n \to X, x_1, \ldots, x_n \in X\} >$.

Let us identify X with a non-trivial finite cyclic group with generating element $g \in X$. Thus we also have that for any $c_1, \ldots, c_n \in X$, ${F^{(1)}}_{\epsilon,j,(c_1,\ldots,c_n)}$, ${F^{(2)}}_{\epsilon,j,(c_1,\ldots,c_n)} \in < \mathcal{T}_{X,n} \cup \{F : X^n \to X^n \mid F(x_1, \ldots, x_n) = (f(x_1, \ldots, x_n), x_2, x_3, \ldots, x_n), f : X^n \to X, x = (x_1, \ldots, x_n) \in X^n\} >$, whenever $\epsilon \in \{1, -1\}$,

${F^{(1)}}_{\epsilon,j,(c_1,\ldots,c_n)}(x) =$

$$\begin{cases} (c_1, \ldots, c_n) & \text{if } x = (c_1, \ldots, c_{j-1}, c_j g^{\epsilon}, c_{j+1}, \ldots, c_n), \\ (c_1, \ldots, c_{j-1}, c_j g^{\epsilon}, c_{j+1}, \ldots, c_n) & \text{if } x = (c_1, \ldots, c_n), \\ x & \text{otherwise,} \end{cases}$$

$${F^{(2)}}_{\epsilon,j,(c_1,\ldots,c_n)}(x) = \begin{cases} (c_1, \ldots, c_{j-1}, c_j g^{\epsilon}, c_{j+1}, \ldots, c_n) & \text{if } x = (c_1, \ldots, c_n), \\ x & \text{otherwise,} \end{cases}$$

where $x = (x_1, \ldots, x_n) \in X^n$. On the other hand, by Lemma 2.4, there exists an arrangement $a_1, \ldots, a_m$ of X^n, such that for every $k = 1, \ldots, m-1$, $p_k \in \{{F^{(1)}}_{\epsilon,j,(c_1,\ldots,c_n)} : \epsilon \in \{-1, 1\}, j \in \{1, \ldots, n\}, c_1, \ldots, c_n \in X\}$, $t_k \in \{{F^{(2)}}_{\epsilon,j,(c_1,\ldots,c_n)} : \epsilon \in \{-1, 1\}, j \in \{1, \ldots, n\}, c_1, \ldots, c_n \in X\}$, where

$$p_k(a_\ell) = \begin{cases} a_{k+1} & \text{if } \ell = k, \\ a_k & \text{if } \ell = k+1, \\ a_\ell & \text{otherwise,} \end{cases}$$

$$t_k(a_\ell) = \begin{cases} a_{k+1} & \text{if } \ell = k, \\ a_\ell & \text{otherwise.} \end{cases}$$

But then $p_1, \ldots, p_{m-1}$ is a set of transpositions such that $\{p_1, \ldots, p_{m-1}\}$ generates all permutations over X^n. And simultaneously, $t_1, \ldots, t_{m-1}$ is a set of elementary collapsings over X^n. Thus by the well-known fact that for every $j = 1, \ldots, m-1$, $\{p_1, \ldots, p_{m-1}, t_j,\}$ generates all transformations over X^n, the proof is complete. □

3 Main Results

First we show the next statement.

Theorem 3.1. *Given a positive integer $n > 1$, $\mathcal{D} = (V, E)$ with $V = \{1, \ldots, n\}$ is an n-complete digraph with minimal number of edges if and only if there exists a permutation $p : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $E = \{(p(i), p(j)) : i, j \in \{1, \ldots, n\}, p(j) = p(i + 1 \bmod n)\} \cup \{(p(i), p(1)) : i \in \{1, \ldots, n\}\}$.*

Proof. We may assume without loss of generality that the permutation p is the identity. Then it is clear that for an arbitrary $m \in \{1,\dots,n\}$, the functions $T^{(0)}, T_\ell^{(k)} : k = 1,2,3,4$ defined in Lemma 2.3 are compatible with $\mathcal{D}$. Suppose that m is 3 such that it is relatively prime to n. Then the sufficiency of this statement is a direct consequence of Lemma 2.5. To the necessity first we show the existence of $j \in V$ with $\{(i,j) : i \in V\} \subseteq E$, whenever $\mathcal{D}$ is n-complete.

Let $T : X^n \to X^n$ such that $|\{T(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}| = |X^n| - 1$. First we show that for every $F_1,\dots,F_m \in \mathcal{T}_{X^n}$, $T = F_1 \circ \dots \circ F_m$ implies the existence of an index i 0 the property $|\{F_i(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}| = |X^n| - 1$. Of course, if $F_1,\dots,F_m$ are injective then $T = F_1 \circ \dots \circ F_m$ should be also injective, a contradiction. On the other hand, $T = F_1 \circ \dots \circ F_m$ implies $|\{F(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}| \leq \min\{|\{F_i(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}| : i = 1,\dots,m\}$. Therefore, we obtain our assumption regarding the existence of an index i preserving the property $|\{F_i(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}| = |X^n| - 1$.

Now we identify the elements of X in a fixed but arbitrary way with the elements of $\{1,\dots,|X|\}$ and consider X^n as a subset of the n^{th} direct power of integers. For every $(a_{1,1},\dots,a_{1,n}),\dots,(a_{m,1},\dots,a_{m,n}) \in X^n$, let $\sum\{(a_{i,1},\dots,a_{i,n}) : i = 1,\dots,m\} = (\sum_{i=1}^m a_{i,1},\dots,\sum_{i=1}^m a_{i,n})$. Let $a = (a_1,\dots,a_n)$, $b = (b_1,\dots,b_n) \in X^n$ denote distinct elements with $|F_i^{-1}(a)| = 0$ and $|F_i^{-1}(b)| = 2$. And let $j \in \{1,\dots,n\}$ be an index with $a_j \neq b_j$.

Prove that $|X|$ does not divide $pr_j(\sum\{F_i(x_1,\dots,x_n) : x_1,\dots,x_n \in X\})$. Indeed, then $pr_j(\sum\{F_i(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}) = pr_j(\sum\{(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}) + b_j - a_j) = |X^{n-1}|(\sum_{k=0}^{|X|-1} k) + b_j - a_j$. Of course, by this equality we received that $|X|$ does not divide $pr_j(\sum\{F_i(x_1,\dots,x_n) : x_1,\dots,x_n \in X\})$.

Suppose that for every $j \in V$ there exists an $i \in V$ with $(i,j) \notin E$. Consider the set $\mathcal{D}_X$ of all functions of the form $X^n \to X^n$ which are compatible with $\mathcal{D}$. Now we show that for every $F \in \mathcal{D}_X$, $|X|$ divides $pr_j(\sum\{F(x_1,\dots,x_n) : x_1,\dots,x_n \in X\})$, implying $F_i \notin \mathcal{D}_X$.

By $F \in \mathcal{D}_X$ we have that for an appropriate $\ell \in \{1,\dots,n\}, pr_j(F(x_1,\dots,x_n)) = pr_j(F(x_1,\dots,x_{\ell-1},x'_\ell,x_{\ell+1},\dots,x_n))$ $((x_1,\dots,x_n) \in X^n, x'_\ell \in X, \ell = j$ is allowed). Therefore, for an arbitrary fixed $c \in X$, $pr_j(\sum\{F(x_1,\dots,x_n) : x_1,\dots,x_n \in X\}) = |X| pr_j(\sum\{F(x_1,\dots,x_{\ell-1},c,x_{\ell+1},\dots,x_n)) : x_1,\dots,x_{\ell-1},x_{\ell+1},\dots,x_n \in X\}$. But then $|X|$ divides $pr_j(\sum\{F(x_1,\dots,x_n) : x_1,\dots,x_n \in X\})$ for every $j = 1,\dots,n$. Hence we get $F_i \notin \mathcal{D}_X$. Consequently, there exists a $T \in \mathcal{T}_{X^n}$ whith $T \notin < \mathcal{D}_X >$. This ends the proof of the existence of $j \in V$ with $\{(i,j) : i \in V\} \subseteq E$, whenever $\mathcal{D}$ is n-complete. Then we are ready if we can prove the existence of a permutation $p : \{1,\dots,n\} \to \{1,\dots,n\}$ having $\{(p(i),p(j)) : i,j \in \{1,\dots,n\}, p(j) = p(i) + 1(\mathrm{mod} n)\} \subseteq E$.

Consider the mapping $T^{(0)} : X^n \to X^n$ defined by $T^{(0)}(x_1,\dots,x_n) = (x_n,x_1,\dots,x_{n-1})$ $(x_1,\dots,x_n \in X)$. To complete the proof of our theorem, we will show $T^{(0)} \notin \mathcal{D}_X$ if there exists no such a permutation p.

It is also clear that an n-complete digraph $\mathcal{D}$, having n vertices, should be strongly connected. Therefore, all vertices have (non-loop) incoming edges. Thus, by the minimality of $|E|$, we get $|E \setminus \{(i,j) : i \in V\}| = n - 1$. Simultaneously,

the strongly connectivity of $\mathcal{D}$ implies $\{j\} \times (V \setminus \{j\}) \cap E \neq \emptyset$ (where $j \in V$ with $\{(i,j) : i \in V\} \subseteq E$). On the other hand, if there exists no permutation p having the above discussed property, then by the strongly connectivity of $\mathcal{D}, V \times \{j\} \subseteq E$ and $|E \setminus \{(i,j) : i \in V\}| = n-1$, we can prove $|\{j\} \times (V \setminus \{j\}) \cap E| \geq 2$, implying the existence of two distinct vertices $i_1, i_2 \in V$ with $\{(\ell, i_r) : r = 1,2, \ell \in V\} \cap E = \{(j,i_1),(j,i_2)\}$.

It is enough to prove that in this case $T^{(0)} \notin \mathcal{D}_X$. Clearly, $F_1 \in \mathcal{D}_X$ implies the existence of functions $f_k : X \to X$, $k = 1,2$ with $pr_{i_k}(F_1(x_1,\ldots,x_n)) = f_k(x_j)$. Therefore, the cardinality of $\{(y_1,y_2) : y_k = pr_{i_k}(F_1(x_1,\ldots,x_n)), k = 1,2,\ x_1,\ldots,x_n \in X\}$ is not greater than $|X|$. In a similar way, for every $F_1,\ldots,F_m \in \mathcal{D}_X, m > 1$ there exist functions $f_k : X \to X, k = 1,2$ such that $pr_{i_k}(F_1 \circ \ldots \circ F_m(x_1,\ldots,x_n)) = f_k(pr_j(F_2 \circ \ldots \circ F_m(x_1,\ldots,x_m)))$ implying that the cardinality of $\{(y_1,y_2) : y_k = pr_{i_k}(F_1 \circ \ldots \circ F_m(x_1,\ldots,x_n)), k = 1,2, x_1,\ldots,x_n \in X\}$ is not greater than $|X|$. On the other side, the cardinality of $\{(y_1,y_2) : y_k = pr_{i_k}(T^{(0)}(x_1,\ldots,x_n)), k = 1,2,\ x_1,\ldots,x_n \in X\}$ is $|X|^2$ yielding to $T^{(0)} \notin \mathcal{D}_X$. The proof is complete. □

Now we prove the following characterization.

Theorem 3.2. *Given a positive integer $n > 1$, $\mathcal{D} = (V,E)$ with $V = \{1,\ldots,m\}$, $m > n$ is an n-complete digraph with minimal number of edges if and only if there exists a permutation $p : \{1,\ldots,m\} \mapsto \{1,\ldots,m\}$ such that $E = \{(p(i),p(j)) : p(i),p(j) \in \{1,\ldots,n+1\}, p(j) = p(i+1 \bmod n+1)\} \cup \{(p(i'),p(j'))\}$, where $i',j' \in \{1,\ldots,n+1\}, |j'-i'| \neq 1$, moreover, $|j'-i'|-1$ and $n+1$ are relatively prime. NB: The case $i' = j'$ is not excluded. Moreover, if there are more than $n+1$ vertices then all except for $n+1$ are isolated.*

Proof. To the sufficiency it is enough to prove for any $n > 2$ the n-completeness of $\mathcal{D} = (\{1,\ldots,n+1\}, \{(i,i+1(\bmod n+1)) : i \in \{1,\ldots,n+1\}\} \cup \{(1,r)\}$, where $r \in \{1,\ldots,n+1\}, r \neq 2$, and in addition, $r-2$ and $n+1$ are relative primes.

Consider the set $\mathcal{D}_X$ of all functions of the form $X^{n+1} \to X^{n+1}$ which are compatible with $\mathcal{D}$. By definition, we obtain $\{T^{(0)}, T_\ell^{(k)} : k = 1,\ldots,4,\} \subsetneq \mathcal{D}_X$, where $T^{(0)}, T_\ell^{(k)}, k = 1,\ldots,4$ are defined as in Lemma 2.3 (taking m of the lemma to be $r-2$). Identifying X with a finite group and using Lemma 2.3, then we get $\mathcal{T}_{X,n} \subsetneq < \mathcal{D}_X >$, too. On the other hand, we have by definition $\{F : X^{n+1} \to X^{n+1} \mid F(x_1,\ldots,x_{n+1}) = (x_{n+1}, x_1, \ldots, x_{r-1}, f(x_1, x_{r-1(\bmod n+1)}), x_{r+1}, \ldots, x_n), f : X^2 \to X, i \in \{1,\ldots,n+1\}, (x_1,\ldots,x_{n+1}) \in X^{n+1}\} \in \mathcal{D}_X$. But then $\{F : X^{n+1} \to X^{n+1} \mid F(x_1,\ldots,x_{n+1}) = (x_1,\ldots,x_{i-1}, f(x_i, x_{i+1(\bmod n+1)}), x_{i+1},\ldots, x_{n+1}), f : X^2 \to X, i \in \{1,\ldots,n+1\}, (x_1,\ldots,x_{n+1}) \in X^{n+1}\} \cup \mathcal{T}_{X,n+1} \subseteq \mathcal{D}_X$ resulting $\Gamma_{X^n} \subseteq \mathcal{D}_X$. Applying Lemma 2.2, this shows the n-completeness of $\mathcal{D}$.

Using the obvious fact that n-complete digraph should have a strongly connected n-complete subdigraph, by our minimality conditions, we will consider digraphs which have a strongly connected subdigraph and all vertices outside of this digraph are isolated. Thus, the sufficiency of our statement implies that by our minimality conditions, we can restrict our investigations to the strongly connected n-complete digraphs having not more than $n+2$ edges. (We can take out

of consideration the isolated vertices.) If we have $n+1$ vertices and fewer than $n+1$ edges then our digraph is not strongly connected. On the other hand, if we consider a strongly connected digraph $\mathcal{D}$ with $n+1$ vertices and $n+1$ edges, i.e., a cycle having $n+1$ length, then for every $F \in < \mathcal{D}_X >$, there exist $k \in \{1,\ldots,n+1\}$, $f_i : X \to X, i = 1,\ldots,n+1$ with $F(x_1,\ldots,x_n) = (f_1(x_k), f_2(x_{k+1(\bmod (n+1))}),\ldots,f_{n+1}(x_{n+k(\bmod n+1)}) \quad (x_1,\ldots,x_n \in X)$. Therefore, for any $1 \leq i_1 < i_2 < \ldots < i_m \leq n+1$, $pr_{i_1,\ldots,i_m}(F(x_1,\ldots,x_{n+1})) = (f_{i_1}(x_{i_1+k(\bmod n+1)}),\ldots,f_{i_m}(x_{i_m+k(\bmod n+1)}))$, $(x_1,\ldots,x_{n+1} \in X)$ which obviously shows that this type of digraphs can not be n-complete.

Therefore, to the necessity of our statement, we can consider only strongly connected digraphs having $n+1$ vertices and $n+2$ edges.

By the strongly connectivity of $\mathcal{D}$ we may suppose that $\mathcal{D} = (V,E)$, with $|V| = n+1, |E| = n+2$, has a cycle $\mathcal{C} = (V',E')$ with k length for some $1 \leq k \leq n+1$, where $V' = \{v_1,\ldots,v_k\}(\subsetneq V)$, $E' = \{(v_i, v_{i+1(\bmod k)}) \mid i = 1,\ldots,k\}(\subsetneq E)$.

Using the strongly connectivity of $\mathcal{D}$ again, for every $V' \subsetneq V$ there are distinct $(v_i,v_j), (v_s,v_t) \in E$ with $v_i, v_t \in V', v_j, v_s \in V \setminus V'$. Therefore, by an induction we get the structure of $\mathcal{D}$ in the following manner.

If $k < n+1$ then $V = \{v_1,\ldots,v_k,v_{k+1},\ldots,v_{n+1}\}, E = E' \cup \{(v_{k+i-1}, v_{k+i}) \mid i = 1,\ldots,n-k+1\} \cup \{(v_{n+1}, v_\ell)\}$, where $\ell \in \{1,\ldots,k\}$ is arbitrarily fixed.

If $k = n+1$ then, of course, $V = V'$, and $E = E' \cup \{(v_{n+1}, v_\ell)\}$ for some $\ell \in \{2,\ldots,n+1\}$.

To complete the case $k = n+1$, first we study digraphs having the form $\mathcal{D} = (\{v_1,\ldots,v_{n+1}\}, \{(v_i, v_{i+1(\bmod n+1)}) : i \in \{1,\ldots,n+1\}\} \cup \{(v_1,v_\ell)\}$, where $\ell \in \{1,\ldots,n+1\}, \ell \neq 2$, such that $\ell - 2 (\bmod n+1)$ and $n+1$ are not relative primes. Then $n+1$ has a divisor $d > 1$ such that for any mapping $F \in \mathcal{D}_X$, $F(x_1,\ldots,x_{n+1}) = (f_1(x_{i_{1,1}},\ldots,x_{i_{1,j_1}}),\ldots, f_{n+1}(x_{i_{n+1,1}},\ldots,x_{i_{n+1,j_{n+1}}})$, where for every $w \in \{1,\ldots,n+1\}, u,v \in \{1,\ldots,j_w\}$, $i_{w,u} \equiv i_{w,v} (\bmod d), i_{w,u} \equiv w - 1 (\bmod d)$ $(x_1,\ldots,x_n \in X)$. These hold for compatible maps, i.e. if $w \neq r$ then f_w depends only on x_{w-1}, otherwise $w = r$ and f_w depends only on x_{r-1} and x_1. It is also clear that every composition of such functions preserves this property. Therefore, for every $F \in < \mathcal{D}_X >$ and $i \in \{1,\ldots,n+1\}$, $pr_i(F)$ depends on proper divisor of $n+1$ many variables which is fewer than n. Therefore, digraphs having this like structures are not n-complete.

It is remained to study the case $k < n+1$. Then $V = \{v_1,\ldots,v_k,v_{k+1},\ldots,v_{n+1}\}$, $E = E' \cup \{(v_{k+i-1}, v_{k+i}) : i = 1,\ldots,n-k+1\} \cup \{(v_{n+1}, v_\ell)\}$, where $\ell \in \{1,\ldots,k\}$ is arbitrarily fixed. Of course, if $k = 1$ or $\ell = 1$ then we have one of the cases discussed previously. Thus we assume $k, \ell \neq 1$.

Given a set X with $|X| \geq 2$, let $\mathcal{M}_X = \{F : X^n \to X^n : |X^n| - 1 \leq |\{F(x_1,\ldots,x_n) : (x_1,\ldots,x_n) \in X^n\}| (\leq |X^n|)\}$. Clearly, then for every $F : X^n \to X^n$, $F \in < \mathcal{M}_X >$.

To complete our proof, now we show that there exists a network $\mathcal{D}' = (V',E')$ with $|V'| = n, E' = V' \times V' \setminus \{(v_i,v_i) : v_i \in V'\}$ such that for every pair $F \in < \mathcal{D}_X >$, $H \subsetneq \{1,\ldots,n+1\}$, $|H| = n$, the existence of $pr_H(F)$ implies $pr_H(F) \in < \mathcal{D}'_X >$ whenever $pr_H(F) \in \mathcal{M}_X$ (where $\mathcal{D}'_X$ denotes the set of all functions of the form

$F : X^n \mapsto X^n$ to be compatible with $\mathcal{D}'$).

Observe that for every $F_m \in \mathcal{D}_X$ there are $f_j : X \to X, j = 1, \ldots, \ell-1, \ell+1, \ldots, n+1, f_\ell : X^2 \to X$ with $F_m(x_1, \ldots, x_n) = (f_1(x_k), f_2(x_1), \ldots, f_{\ell-1}(x_{\ell-2}), f_\ell(x_{\ell-1}, x_{n+1}), f_{\ell+1}(x_\ell), \ldots, f_{n+1}(x_n))((x_1, \ldots, x_{n+1}) \in X^{n+1})$. Therefore, $H = \{1, \ldots, n+1\}\backslash\{i\}$, $i \in \{1, \ldots, n+1\}\backslash\{\ell-1, n+1\}$ and $F = F_1 \circ \ldots \circ F_m, F_1, \ldots, F_m \in \mathcal{D}_X$ implies $|\{pr_H(F)(x_1, \ldots, x_n) : (x_1, \ldots, x_n) \in X^n\}| \leq |X^{n-1}|$. Hence, in this case $pr_H(F) \notin \mathcal{M}_X$. Thus we may assume $H = \{1, \ldots, n+1\}\backslash\{i\}$, $i \in \{\ell-1, n+1\}$. In addition, it is clear that by the structure of $\mathcal{D}$, for every $T \in \mathcal{D}_X$, $cp_1(T)$ and $cp_{k+1}(T)$ may really depend only on the same k^{th} variable of F.

Let $F = F_1 \circ \ldots \circ F_m$ with $F_1, \ldots, F_m \in \mathcal{D}_X$, such that, $pr_H(F) \in \mathcal{M}_X$ exists for a suitable $H = \{1, \ldots, i-1, i+1, \ldots, n+1\}$, $i \in \{\ell-1, n+1\}$.

First we suppose $m = 1$. Consider functions $f_j : X \to X, j \in \{1, \ldots, \ell-1, \ell+1, \ldots, n+1\}, f_\ell : X^2 \to X$ with $F(x_1, \ldots, x_{n+1}) = (f_1(x_k), f_2(x_1), \ldots, f_{\ell-1}(x_{\ell-2}), f_\ell(x_{\ell-1}, x_{n+1}), f_{\ell+1}(x_\ell), \ldots, f_{n+1}(x_n))$ $(x_1, \ldots, x_{n+1}) \in X^{n+1})$. Clearly, then $i \in \{1, k+1\}$ also holds provided $pr_H(F) \in \mathcal{M}_X$.

Suppose $i = 1$. Then in consequence of $i \in \{\ell-1, n+1\}$, we have $\ell = 2$. Clearly, then f_2 really may not depend on its first variable, i.e. there exists a $g : X \to X$ with $f_2(x_1, x_2) = g(x_2)$ $(x_1, x_2 \in X)$. Construct the function $T : X^n \to X^n$ with $T(x_1, \ldots, x_n) = (g(x_n), f_3(x_1), \ldots, f_{n+1}(x_{n-1}))$ $((x_1, \ldots, x_n) \in X^n)$. Then we get $pr_H(F) = T$. On the other side, $T \in \mathcal{D}'_X$ is also obvious.

Suppose $i = k+1$. By $i \in \{\ell-1, n+1\}$ and $\ell \leq k$, this implies $k = n$. On the other side, then f_ℓ really may not depend on its second variable, i.e. there exists a $g : X \to X$ with $f_\ell(x_1, x_2) = g(x_1)$ $(x_1, x_2 \in X)$: Let $T : X^n \to X^n$ with $T(x_1, \ldots, x_n) = (f_1(x_n), f_2(x_1), \ldots, f_{\ell-1}(x_{\ell-2}), g(x_{\ell-1}), f_{\ell+1}(x_\ell), \ldots, f_n(x_{n-1})$ $((x_1, \ldots, x_n) \in X^n)$. It is obvious that $T \in \mathcal{D}'_X$ and $pr_H(F) = T$.

Now we turn to the case $m > 1$. Then first we define the mappings $F'_1, \ldots, F'_m \in \mathcal{D}_X$ in the following way. For every $r = 1, \ldots, m$, define functions $f_r : X \mapsto X, g_r : X \mapsto X$ with $f_r(x) = pr_1(F_r(x_1, \ldots, x_{k-1}, x, x_{k+1}, \ldots, x_{n+1}))$, $g_r(x) = pr_{k+1}(F_r(x_1, \ldots, x_{k-1}, x, x_{k+1}, \ldots, x_{n+1}))$, $x_1, \ldots, x_{k-1}, x, x_{k+1}, \ldots, x_{n+1} \in X$. ($F_r \in \mathcal{D}_X$ implies that f_r and g_r are well-defined.) In addition, let for every $r = 1, \ldots, m$, $pr_j(F'_r(x_1, \ldots, x_{n+1})) =$

$$\begin{cases} f_1(x_k) & \text{if } r = 1 \text{ and } j = 1, \\ g_1(x_k) & \text{if } r = 1 \text{ and } j = k+1, \\ x_k & \text{if } r > 1 \text{ and } j \in \{1, k+1\}, \\ pr_j(F_m(x_1, \ldots, x_{n+1})) & \text{if } r = m \text{ and } j \in \{2, \ldots, k, k+2, \ldots, n+1\}, \\ pr_j(F_r(f_{r+1}(x_1), x_2, \ldots, x_k, & \\ \quad g_{r+1}(x_{k+1}), x_{k+2}, \ldots, x_{n+1})) & \text{otherwise} \end{cases}$$

$(x_1, \ldots, x_{n+1} \in X)$. By an easy computation we get $F_1 \circ \ldots \circ F_m = F'_1 \circ \ldots \circ F'_m$. Define for a fixed $c \in X$, $m = 2$, $pr_j(F''_r(x_1, \ldots, x_n))$

$$= \begin{cases} pr_j(F'_r(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } 1 \leq j < i, \\ pr_{j+1}(F'_r(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } i \leq j \leq n \end{cases}$$

$(x_1, \ldots, x_{n+1} \in X, r = 1, 2)$.

Similarly, for a fixed $c \in X$ and $m = 3$, let $pr_j(F''_r(x_1, \ldots, x_n))$

$$= \begin{cases} pr_j(F'_1(x_k, x_2, \ldots, x_{\ell-2}, x_n, x_{\ell-1}, \ldots, x_{n-1}, x_1)) & \text{if } i = \ell - 1, r = 1 \text{ and } 1 \le j < \ell - 1, \\ pr_{j+1}(F'_1(x_k, x_2, \ldots, x_{\ell-2}, x_n, x_{\ell-1}, \ldots, x_{n-1}, x_1)) & \text{if } i = \ell - 1, r = 1 \text{ and } \ell - 1 \le j \le n, \\ pr_j(F'_1(x_{k+1}, x_2, \ldots, x_n)) & \text{if } i = n + 1, r = 1 \text{ and } 1 \le j \le n, \\ pr_{n+1}(F'_2(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = 2 \text{ and } j = 1, \\ pr_j(F'_2(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = 2 \text{ and } 1 < j < i, \\ pr_{j+1}(F'_2(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = 2 \text{ and } i \le j \le n, \\ pr_j(F'_3(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = 3 \text{ and } 1 \le j < i, \\ pr_{j+1}(F'_3(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = 3 \text{ and } i \le j \le n, \end{cases}$$

$(x_1, \ldots, x_{n+1} \in X, r = 1, 2, 3)$.

In addition, let for a fixed $c \in X$ and $m > 3$, $pr_j(F''_r(x_1, \ldots, x_n))$

$$= \begin{cases} pr_j(F'_r(x_k, x_2, \ldots, x_{\ell-2}, x_n, x_{\ell-1}, \ldots, x_{n-1}, x_1)) & \text{if } i = \ell - 1, r = 1, 1 \le j < \ell - 1, \text{ or } i = \ell - 1, 1 < r \le m - 2 \text{ and } 1 < j < \ell - 1, \\ pr_{j+1}(F'_r(x_k, x_2, \ldots, x_{\ell-2}, x_n, x_{\ell-1}, \ldots, x_{n-1}, x_1)) & \text{if } i = \ell - 1, r = 1, \ell - 1 \le j \le n, \text{ or } i = \ell - 1, 1 < r \le m - 2 \text{ and } \ell - 1 \le j \le n, \\ pr_{n+1}(F'_r(x_k, x_2, \ldots, x_{\ell-2}, x_n, x_{\ell-1}, \ldots, x_{n-1}, x_1)) & \text{if } i = \ell - 1, 1 < r \le m - 2 \text{ and } j = 1, \\ pr_{n+1}(F'_r(x_{k+1}, x_2, \ldots, x_n)) & \text{if } i = n + 1, 1 < r \le m - 2 \text{ and } j = 1, \\ pr_j(F'_r(x_{k+1}, x_2, \ldots, x_n)) & \text{if } i = n + 1, r = 1, 1 \le j \le n, \text{ or } i = n + 1, 1 < r \le m - 2 \text{ and } 1 < j \le n, \\ pr_{n+1}(F'_{m-1}(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = m - 1 \text{ and } j = 1, \\ pr_j(F'_{m-1}(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = m - 1 \text{ and } 1 < j < i, \\ pr_{j+1}(F'_{m-1}(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = m - 1 \text{ and } i \le j \le n, \\ pr_j(F'_m(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = m \text{ and } 1 \le j < i, \\ pr_{j+1}(F'_m(x_1, \ldots, x_{i-1}, c, x_i, \ldots, x_n)) & \text{if } r = m \text{ and } i \le j \le n, \end{cases}$$

$(x_1, \ldots, x_{n+1} \in X, r \in \{1, \ldots, n\}))$.

We remark that, of course, for every $j = 2, \ldots, m$, the value of $F''_j \circ \ldots \circ F''_m(x_1, \ldots, x_m)$ $(x_1, \ldots, x_n \in X)$ may depend of the value of (the above fixed) $c \in X$. But the value of $F''_1 \circ \ldots \circ F''_m(x_1, \ldots, x_m)$ $(x_1, \ldots, x_n \in X)$ may not depend on the value of $c \in X$ in question, because $F_H = F''_1 \circ \ldots \circ F''_m$ by definition.

(Remember that the existence of $pr_H(F)$ $(= pr_H(F_1 \circ \ldots \circ F_m), m > 1)$ is supposed with $H = \{1, \ldots, i-1, i+1, \ldots, n+1\}$ for a fixed $i \in \{\ell - 1, n+1\}$.)

By an elementary computation we can prove $F_1'', \ldots, F_m'' \in \mathcal{D}'_X$. Applying Theorem 3.1, $\mathcal{D}'$ may not be n-complete because it is not centralized. Therefore, there exists a $T \in \mathcal{M}_X$ with $T \notin < \mathcal{D}'_X >$. But then for every $F \in < \mathcal{D}_X >$, $H = \{1, \ldots, n+1\}$, $|H| = n$, $pr_H(F) \neq T$. Therefore, $\mathcal{D}$ can not be n-complete.

This ends the proof. □

References

[1] Dömösi, P. and Kovács, B., Simulation on finite networks of automata. *Words, Languages and Combinatorics, Ed. by M. Ito (Kyoto, 1990), 131-138, World Sci. Publishing, River Edge, NJ, 1992.*

[2] Dömösi, P., Nehaniv, C. L., Some Results and Problems on Finite Homogeneous Automata Networks, Proc. Japanese Association of Mathematical Sciences Annual Meeting on "Languages, Computation and Algebra", Kobe University, August 27-28, 1997 (in press).

[3] Tchuente, M., Computation on Finite Networks of Automata, *In: Automata Networks, Ed. by C. Choffrut (Argelès-Village, France, 1986), 53-67, Lecture Notes in Computer Science 316, 1988.*

Acta Cybernetica **14** (1999) 51-64.

Trips on Trees

Joost Engelfriet* Hendrik Jan Hoogeboom*
Jan-Pascal Van Best *

Abstract

A "trip" is a triple (g, u, v) where g is, in general, a graph and u and v are nodes of that graph. The trip is from u to v on the graph g. For the special case that g is a tree (or even a string) we investigate ways of specifying and implementing sets of trips. The main result is that a regular set of trips, specified as a regular tree language, can be implemented by a tree-walking automaton that uses marbles and one pebble.

1 Introduction

A specification of a function describes the result of the function in terms of its argument. The goal of the programmer is to implement this specification by a program that, for a given argument as input, produces the function result as output. From an elementary point of view, the program can be seen as a device that walks on a graph g. The nodes of g are the possible contents of the program variables, and there is an edge from node m to node n if m can be transformed into n by an atomic programming statement, such as an assignment. The program should find its way through this graph from the initial state u, determined by the input, to the final state v, that determines the output.

In this paper we consider a special case of this general situation, viz. the case that the graph is a finite tree (or even a finite string). In particular, the specification describes a set of triples of the form (t, u, v) where t is a tree (over a ranked alphabet) and u and v are nodes of t. Each such triple can be viewed as a "trip" from u to v on the tree t. Thus, the specification describes a set of trips, i.e., a "trip type". To simplify terminology we will also call this a trip. An example of a trip (type) is: from the left-most leaf to the right-most leaf. This is, of course, the set of all triples (t, u, v) where t is an arbitrary tree, u is its left-most leaf, and v is its right-most leaf.

A set of triples (t, u, v) is said to be *regular* if it forms a regular tree language when, as is quite usual, the nodes u and v are indicated by special marks in t. Thus, a regular trip can be specified by a finite tree automaton (that recognizes

*Department of Computer Science, Leiden University P.O.Box 9512, 2300 RA Leiden, The Netherlands e-mail: engelfri@wi.leidenuniv.nl

the trip) or a regular tree grammar (that generates the trip) or a regular expression (that describes the trip). Moreover, as is well known from [Don, ThaWri] (and from [Büc, Elg] for strings), it can be specified by a formula $\phi(u, v)$ of monadic second-order logic (on trees), with two free variables u and v. Thus, monadic second-order logic is the highest-level specification language for regular trips.

Our main interest is the implementation of regular trip specifications. Given such a specification of trips (t, u, v), we wish to know how we can walk from u to v on t. In other words, we are looking for a general type of automaton that, when started at node u of t can walk to node v along the edges of the tree. Thus, for the trip mentioned above, the automaton should be able to walk to the right-most leaf, whenever it is "dropped" at the left-most leaf (and go into a rejecting state when dropped at any other node).

It is known from [Blo, BloEng2] that, in general, this cannot be done by a finite state tree-walking automaton (as used, in the form of 'routing expressions', in [KlaSch] to specify data types consisting of trees with additional pointers). The solution in [Blo, BloEng2, BloEng1] is to equip the finite state tree-walking automaton with more powerful tests; in fact, it is allowed to test any property of its current node that can be expressed by a formula (with one free variable) of monadic second-order logic. This is of course not a complete implementation because the tests are still specified in logic. Thus, the question remained how these tests can be implemented. Here we show that regular trips can be implemented by a finite state tree-walking automaton that uses "marbles" and one pebble to find its way through the tree (as Tom Thumb through the forest). The precise way of using marbles and pebble will be explained in Section 4. In Section 3 we start with the easier case of regular trips on strings, and show how to implement them by 2-way finite state automata with one pebble (and no marbles). Section 2 contains the formal definition of a trip.

The results of this paper are part of the Master's Thesis [vBest] of the last author, where more detailed definitions and proofs can be found.

2 Trips and Sites

It should be clear that the reader is assumed to be familiar with formal language theory, and in particular with tree language theory. Thus, the notions of regular tree language and (bottom-up) finite tree automaton are assumed to be good friends of the reader. This can be accomplished by reading [GécSte1] and [GécSte2]. Shame on the reader if he/she did not do so yet!

As explained in the introduction, we are interested in trips on trees. These are now formally defined. Let Σ be a ranked alphabet.

Definition 2.1 *A* trip *is a set of triples (t, u, v) where t is a tree over Σ, and u and v are nodes of t.*

Trips go from sites to sites (or from sights to sights?). This is an auxiliary notion that we will need too.

Definition 2.2 *A* site *is a set of pairs* (t, u) *where* t *is a tree over* Σ *and* u *is a node of* t.

To define regular trips (and sites) we first have to show how nodes of trees can be marked. As usual, to code (t, u, v), two booleans are added to the labels of t, one for u and one for v. We define the "marked" ranked alphabet $\mathrm{m}(\Sigma) = \{(\sigma, b_1, b_2) \mid \sigma \in \Sigma, b_1, b_2 \in \{0, 1\}\}$, where (σ, b_1, b_2) has the same rank as σ. We identify $(\sigma, 0, 0)$ with σ; thus, for each $\sigma \in \Sigma$, $\mathrm{m}(\Sigma)$ contains the symbols σ, $(\sigma, 1, 0)$, $(\sigma, 0, 1)$, and $(\sigma, 1, 1)$. Let t be a tree over Σ and let u and v be nodes of t. We define $\mathrm{mark}(t, u, v)$ to be the tree over $\mathrm{m}(\Sigma)$, with the same nodes and edges as t but with different node labels: if node x has label σ in t, then it has label $(\sigma, x = u, x = v)$ in $\mathrm{mark}(t, u, v)$. Thus, in $\mathrm{mark}(t, u, v)$, either $u \neq v$ and u is marked by $(1, 0)$ and v by $(0, 1)$, or $u = v$ and it is marked by $(1, 1)$; the other nodes are not marked. This defines the coding of trips as tree languages. To code sites as tree languages, we define $\mathrm{mark}(t, u) = \mathrm{mark}(t, u, u)$. Thus, for technical convenience, a site (t, u) gets the same encoding as the "round-trip" (t, u, u). As usual, for a trip T, we define $\mathrm{mark}(T) = \{\mathrm{mark}(t, u, v) \mid (t, u, v) \in T\}$, and for a site S, $\mathrm{mark}(S) = \{\mathrm{mark}(t, u) \mid (t, u) \in S\}$.

Definition 2.3 *A trip* T *is* regular *if* $\mathrm{mark}(T)$ *is a regular tree language. A site* S *is* regular *if* $\mathrm{mark}(S)$ *is a regular tree language.*

As observed in the Introduction, it is well known from the classical results of [Don, ThaWri] that a trip T is regular iff it can be defined by a formula $\phi(x, y)$ of monadic second-order logic, in the sense that T is the set of all (t, u, v) such that $t \models \phi(u, v)$. And similarly for sites and formulas $\phi(x)$ with one free variable.

We will be interested in particular in functional trips, i.e., trips in which the destination is determined by the place of departure. We will show that functional trips can be implemented by deterministic tree-walking automata. It is easy to see that functionality is decidable for regular trips.

Definition 2.4 *A trip* T *is* functional *if there are no triples* $(t, u, v_1), (t, u, v_2) \in T$ *with* $v_1 \neq v_2$.

Finally, all the above definitions also apply to the case of strings over an (ordinary) alphabet Σ, with the appropriate changes. Thus, a trip on strings is a set of triples (w, u, v) with $w \in \Sigma^*$ and u, v are positions of w (i.e., $1 \leq u, v \leq |w|$, where $|w|$ is the length of w). Note that w cannot be the empty string.

3 Trips on Strings

To find our way on strings we will use 2-way pebble automata. A *2-way pebble automaton* is an ordinary 2-way (nondeterministic) finite state automaton with one pebble. The input string is surrounded by endmarkers on the input tape, and at each moment the automaton is at a certain cell of the tape, in a certain state. It

can test the input symbol at the current cell, and move one cell to the left or right, changing state. Additionally, it can drop the pebble on the current cell, it can test whether the pebble is at the current cell, and it can lift the pebble from the current cell (when it lies there, of course). Initially the pebble is not on the input tape, and it is also required that at the end of a computation the pebble is not on the input tape. A 2-way pebble automaton A recognizes a language $L(A)$ in the usual way, but we want to use it to compute a trip, as follows. The *trip* $T(A)$ *computed by* A consists of all triples (w, u, v) such that when A is started at position u of w on the input tape, in its initial state, it can walk to position v, enter a final state, and halt. In other words, if you want to make trip $T(A)$, catch automaton A!

It is well known that 2-way finite automata (without pebble) recognize the regular languages [RabSco, She]. In fact, a 2-way finite automaton A can be simulated by an ordinary (1-way) finite automaton M that, at each cell, computes the *transition table* of A, i.e., the finite set of pairs (q, q') such that when A is started in state q at this cell, it can make an excursion to the left, and return to the cell in state q'. The same technique of transition tables can be used to show that also 2-way pebble automata recognize the regular languages ([BluHew]; cf. [Bir, GloHar] and Exercise 3.19 of [HopUll]): a 2-way pebble automaton A can be simulated by a 2-way automaton A' (without pebble) that at each cell computes two transition tables of the automaton A (without the instructions that manipulate the pebble), one for excursions to the left and one for excursions to the right. Instead of dropping a pebble on a cell, making excursions to the left and right, and then lifting the pebble again, A' can just stay at the cell and compute A's state change from the two transition tables.

From this it is easy to see that every trip computed by a 2-way pebble automaton is regular.

Lemma 3.1 *For every 2-way pebble automaton A, $T(A)$ is a regular trip.*

Proof. We have to show that $\mathrm{mark}(T(A)) = \{\mathrm{mark}(w, u, v) \mid (w, u, v) \in T(A)\}$ is a regular language. In fact, there is a 2-way pebble automaton A' that recognizes $\mathrm{mark}(T(A))$, i.e., $L(A') = \mathrm{mark}(T(A))$. The automaton A' first walks to u (which is marked by $(1,0)$ or $(1,1)$), then simulates a successful walk of A (ignoring marks), and finally checks that it is at v (which is marked by $(0,1)$ or $(1,1)$). □

Determinism of 2-way pebble automata is defined in the usual way. It should be clear that the trip $T(A)$ computed by a deterministic 2-way pebble automaton A is functional (cf. Definition 2.4). We now prove that every regular trip can be computed by a 2-way pebble automaton, and in particular by a deterministic automaton if the trip is functional.

Lemma 3.2 *For every regular trip T on strings there is a 2-way pebble automaton A with $T(A) = T$. Moreover, if T is functional, then A is deterministic.*

Proof. Let M be an ordinary, deterministic finite automaton that recognizes $\mathrm{mark}(T)$. First we describe a nondeterministic automaton A that computes T. The automaton A is started at position u of string w, and it has to walk to position

v, with $(w,u,v) \in T$. To do this, A first guesses whether v is to the left or to the right of u, or $v = u$. Suppose that it guesses v to be to the right of u. A drops the pebble at the start position u, walks to the head of the input tape, and then simulates M walking to the right, until it detects the pebble. It picks up the pebble and continues the simulation of M, treating the symbol σ at position u as $(\sigma,1,0)$. Then, nondeterministically, A drops the pebble at some position v, treats the symbol σ at position v as $(\sigma,0,1)$, and continues the simulation of M until it reaches the end of the input tape. If M is in a final state, A backs up until it finds the pebble at position v, lifts the pebble, and goes into a final state. In the case that v is to the left of u, A simulates a deterministic finite automaton that recognizes the mirror image of mark(T), walking from the end to the beginning of the input tape. The case that $v = u$ is obvious.

Let us now assume that T is functional, and describe a deterministic automaton A. It is a variation of the nondeterministic automaton A above. First we argue that A can find out deterministically whether v is to the left or right of u, or at u. Since mark(T) is a regular language, it should be clear that the language $\{\text{mark}(w,u,v) \mid (w,u,v) \in T \text{ and } v \text{ is to the right of } u\}$ is regular too. Hence, applying the string homomorphism that changes $(\sigma,1,0)$ into $(\sigma,1,1)$, and $(\sigma,0,1)$ into σ, to this language, we obtain that the site $S = \{(w,u) \mid (w,u,v) \in T \text{ for some } v \text{ to the right of } u\}$ is regular. Thus, A can test whether or not v is to the right of u by testing whether or not (w,u) is in site S, and it can do that by dropping its pebble at u and simulating a deterministic finite automaton that recognizes mark(S), treating the symbol σ at position u as $(\sigma,1,1)$. Obviously, A can test in a similar way whether or not v is to the left of u, or at u. Suppose now that v is to the right of u. A then behaves as in the nondeterministic case, simulating M, until it picks up the pebble from u. After that, instead of guessing v nondeterministically, A just tries out all positions v to the right of u, one by one from left to right, moving its pebble from one v to the next. Note that, when walking from v to the end of the tape, A should not only keep track of the current state of M but also remember the state in which M arrived in v; this allows A to continue the simulation of M with the next v. □

Altogether we have proved that the 2-way pebble automaton is the implementation model of regular trips on strings.

Theorem 3.3 *A trip on strings is regular iff it can be computed by a 2-way pebble automaton. A functional trip on strings is regular iff it can be computed by a deterministic 2-way pebble automaton.*

Since a trip is regular iff it can be expressed in monadic second-order logic, this theorem can be viewed as the generalization from languages to trips of the classical result of Büchi and Elgot [Büc, Elg].

It is shown in [Blo, BloEng2] (for the more general case of trees) that 2-way finite automata cannot compute all regular trips. Thus, the pebble is really needed. We strengthen this result in Theorem 4.9. On the other hand, it is well known that two pebbles are more powerful than one; a 2-way automaton with two pebbles can

easily recognize, e.g., the language $\{wcw \mid w \in \{a,b\}^*\}$, and thus also compute non-regular trips.

4 Tree-Walking Automata

In the case of strings we have used 2-way automata to walk from one position of a string to another. For trees we need an automaton that walks from one node of a tree to another. Such tree-walking automata were introduced in [AhoUll], and were studied, e.g., in [ERS, KamSlu]. A (nondeterministic) finite state *tree-walking automaton* (or tw automaton, for short) is similar to a 2-way automaton on strings. At each moment the tw automaton is at a certain node of the input tree, in a certain state. It can test the label of the current node, and move to the parent or to one of the children of the node, changing state. A child can be specified by a number between 1 and the rank of the current node label. The automaton can also test whether the current node is the root of the input tree, and if not, what is its "child number", i.e., which child it is of its parent (specified by a number between 1 and the rank of the label of its parent). The language $L(A)$ recognized by a tree-walking automaton A consists of all trees on which A has a computation that starts at the root of the input tree in its initial state, and ends in a final state. As in the case of strings, it can be shown, using the technique of transition tables (for excursions in a subtree), that every tree-walking automaton can be simulated by an ordinary (bottom-up) finite tree automaton. However, as opposed to the case of strings, it is not known whether every finite tree automaton can be simulated by a tree-walking automaton!

Conjecture 4.1 *The class of tree languages recognized by tree-walking automata is a proper subclass of the regular tree languages.*

It should be mentioned here that a statement similar to the one above is proved in [KamSlu]. However, the tree-walking automata of [KamSlu] are weaker than ours: they cannot test the child number of a node; and for this reason, as shown in [KamSlu], they cannot even make a depth-first left-to-right search of the input tree.

Clearly, a type of automaton that can compute all regular trips on trees, should be able to recognize the regular tree languages: for every regular tree language L, $\{(t,u,u) \mid t \in L, u \text{ is the root of } t\}$ is a regular trip, and obviously, an automaton that computes this trip also recognizes L. Thus we are led to an automaton that is known to recognize the regular tree languages: the tree-walking marble automaton.

A *tree-walking marble automaton* is a tree-walking automaton that, additionally, can use "marbles" to drop on the nodes of the input tree. The difference between a pebble and a marble is that the automaton has an unlimited supply of marbles (i.e., a marbles bag of infinite size!). Moreover, we want our automaton to have marbles of different colours (which is the reason to call them marbles). Thus, each automaton has a fixed (but arbitrary) number of marble colours, and it has an

unlimited supply of marbles of each colour. During its computation, the automaton can drop a marble of a given colour on the current node (provided there is not yet one of that colour), it can test whether a marble of a given colour is at the current node, and it can lift a marble of a given colour from the current node (provided there is one there). Note that there cannot be two marbles of the same colour on a node. There is, however, an important additional restriction on the behaviour of the tw marble automaton: if there are marbles on the current node, then the automaton is not allowed to move up to the parent node. In other words, dropping a marble on a node u closes off the context of u, in the sense that the automaton can only visit u and its descendants, but has to lift all marbles from u to visit the other nodes. Since the automaton starts its computation without marbles on the input tree, this restriction implies that at each moment of time all marbles lie on the path from the current node to the root.

It is shown in [KamSlu] (cf. also [ERS]) that the tw marble automaton recognizes exactly the regular tree languages. However, the model of tw marble automaton is described in a different way in these papers. Instead of marbles, the tree-walking automaton has a pushdown, which has the same length as the path from the current node to the root. The pushdown is synchronized with the movements of the automaton on the tree: a symbol is pushed on the pushdown when the automaton moves to a child, and the top symbol is popped from the pushdown when the automaton moves to the parent. It should be clear that these two types of automata recognize the same tree languages. Each symbol on the pushdown can be simulated by a marble on the corresponding node, taking all pushdown symbols as marble colours. Vice versa, the marbles on the path from the current node to the root can be simulated by a pushdown containing in each cell the colours of the marbles that are on the corresponding node, taking all sets of marble colours as pushdown symbols. The only reason that we have turned the tree-walking pushdown automaton into a tree-walking marble automaton is that the pushdown automaton is not suitable for the computation of trips: when started at a node of the input tree, what would be the content of its pushdown?

The result of [KamSlu] is stated next, together with a sketch of the proof.

Proposition 4.2 *Both the nondeterministic and the deterministic tw marble automata recognize the regular tree languages.*

Proof. The fact that the language recognized by a nondeterministic tw marble automaton is regular can be proved in the usual way using transition tables, and we will not go into that (cf. the discussion before Lemma 3.1).

The other way around we sketch how a deterministic tw marble automaton A can simulate a (deterministic, bottom-up) finite tree automaton M. Let us assume for convenience that the input trees are binary, i.e., that the rank of an input symbol is either 2 or 0. A traverses the input tree t in a depth-first left-to-right fashion, and uses the states of M as marble colours. At each node u of t it determines the state in which M arrives at u (in its own state), as follows. If u is a leaf, it determines M's state from the transition function of M. Otherwise, suppose it has determined

the state q_1 at the first child of u. It then drops a marble of colour q_1 on u, walks down to the second child of u, and determines the state q_2 at that child. Moving up to u again, it picks up the marble q_1, and determines the state at u from q_1 and q_2, using M's transition function.

For arbitrary input trees, A uses as marble colours all pairs (i, q), indicating that q is the state of M at the i-th child of u. □

As in the case of strings, to obtain an implementation model for the regular trips an additional pebble is needed. This finally leads us to the main automaton model of this paper: the tree-walking marble/pebble automaton. A *tree-walking marble/pebble automaton* is a tw marble automaton that uses one additional pebble. The pebble can be dropped on a node, detected at a node, and lifted from a node, as usual. Initially and finally, there are no marbles and no pebble on the input tree. However, we need an additional restriction on the behaviour of this automaton, because otherwise non-regular tree languages could be recognized (and hence non-regular trips computed).

Example 4.3 *Consider the non-regular monadic tree language* $\{a^n c b^n e \mid n \geq 0\}$, *with* a, b, c *of rank 1 and* e *of rank 0. This language can be recognized as follows, using the pebble and just one marble: put the marble at the root, and the pebble at the lowest* b*; then move the marble one node down, and the pebble one node up; repeat this last step, until both the marble and the pebble are at the* c*-labeled node.*

The additional restriction on the tw marble/pebble automaton is: the pebble can only be dropped or lifted when there are no marbles on the tree. Note that the automaton is able to keep track of this condition by giving a special colour to the first marble it drops on the tree. At the end of this section we will discuss a less restrictive definition.

The *trip* $T(A)$ *computed by* a tw marble/pebble automaton A is defined just as in the case of 2-way pebble automata on strings: $T(A)$ consists of all triples (t, u, v) such that when A is started at node u of input tree t, in its initial state, it can walk to node v, enter a final state, and halt. So, for this trip you have to catch marble/pebble automaton A!

We first want to prove that every trip computed by a tw marble/pebble automaton is regular. As in the case of strings (cf. Lemma 3.1), this easily follows from the fact that the tree languages recognized by tw marble/pebble automata are regular, i.e., that the above restriction has been effective.

We need some terminology on finite tree automata. Let M be a (deterministic, bottom-up) finite tree automaton, and let u be a node of an input tree t. By $\text{state}_{M,t}(u)$ we denote the state in which M arrives at node u. By $\text{succ}_{M,t}(u)$ we denote the set of states q of M such that M arrives in a final state at the root of t when it is assumed to be in state q at node u (and thus skips the processing of the subtree with root u); such a state q is said to be "successful" at u. Note that, for every node u, $t \in L(M)$ iff $\text{state}_{M,t}(u) \in \text{succ}_{M,t}(u)$. Note also that, for a child v of u, $\text{succ}_{M,t}(v)$ can be determined, using M's transition function (for the label of u), from $\text{succ}_{M,t}(u)$ and the states $\text{state}_{M,t}(v')$ for all children $v' \neq v$ of u: to

determine whether state p is successful at v, one applies M's transition function to p and all $\text{state}_{M,t}(v')$, and checks whether the resulting state q is successful at u.

Theorem 4.4 *The tw marble/pebble automata recognize the regular tree languages.*

Proof. By Proposition 4.2 it suffices to show that every tw marble/pebble automaton A can be simulated by a tw marble automaton A'. Consider a part of a computation of A which starts by dropping the pebble on node u of t, in state q, and ends by lifting it again from u, in state q'. Note that at both moments there are no marbles on t. When simulating A, A' can of course not put a pebble on u (and a marble would not help because it closes off the context of u). Instead it should, somehow, test whether A can make one of the excursions described above, where q is the current state of A and q' is any state of A. In other words, it should be able to test whether (t,u) is in the site $S_{q,q'}$ that consists of all pairs (t,u) such that A can make the excursion described above. We first observe that the site $S_{q,q'}$ is regular. In fact, it is quite clear that $\text{mark}(S_{q,q'})$ can be recognized by a tw marble automaton (and hence is regular by Proposition 4.2): the automaton first walks to node u which is marked by $(1,1)$, and then simulates A, starting in state q, treating the mark $(1,1)$ as the pebble of A, and ending in state q' at u.

Thus, it now suffices to show that a tw marble automaton A' can always be modified in such a way that it can, at each moment, test whether its current node belongs to a given site S. Moreover, the test should be done in a deterministic way because, in the simulation above, several of these sites have to be tested sequentially, viz. $S_{q,q'}$ for all q'. Let M be a bottom-up finite tree automaton that recognizes $\text{mark}(S)$. Note that the input alphabet of M is $\text{m}(\Sigma)$ where Σ is the input alphabet of A'. Clearly, at node u of t, A' can always compute $\text{state}_{M,t}(u)$, using the procedure described in the proof of Proposition 4.2 (first dropping a marble with a special colour on u, to recognize it after traversing the subtree). Let u have label σ of rank k. To determine whether (t,u) is in S, A' visits the children $u_1,\ldots,u_k$ of u, computes $\text{state}_{M,t}(u_i)$ for every $1 \leq i \leq k$, and returns to u. It then computes $q = \text{state}_{M,\text{mark}(t,u)}(u)$, using M's transition function for the symbol $(\sigma,1,1)$. Finally, it checks whether $q \in \text{succ}_{M,t}(u)$. Thus, it remains to explain how the latter test can be implemented. During its computation, A keeps track of $\text{succ}_{M,t}(u)$ by using additional marbles that have the sets of states of M as colours; in particular, there is a marble with colour $\text{succ}_{M,t}(u')$ on every node u' on the path from the current node to the root of t. When A moves from a node u to one of its children v, it can compute $\text{succ}_{M,t}(v)$ (i.e., the colour of the new marble) as described just before this theorem, from $\text{succ}_{M,t}(u)$ (the colour of the marble at u) and the states of M at the other children $v' \neq v$ of u (which it can compute as shown above). When A moves up to the parent of u, it of course first lifts the "succ-marble" from u. Initially A puts a succ-marble with colour F on the root of t, where F is the set of final states of M. □

It is now easy to prove the analogue of Lemma 3.1 for trees.

Lemma 4.5 *For every tw marble/pebble automaton A, $T(A)$ is a regular trip.*

Proof. By the previous theorem it suffices to show that there is a tw marble/pebble automaton A' that recognizes $\mathrm{mark}(T(A))$. Just as in the proof of Lemma 3.1, A' walks to u, simulates A, and checks that it is at v. □

Next we prove that regular trips can be computed by tw marble/pebble automata.

Lemma 4.6 *For every regular trip T on trees there is a tw marble/pebble automaton A with $T(A) = T$. Moreover, if T is functional, then A is deterministic.*

Proof. It is shown in [Blo, BloEng1, BloEng2] (using terminology from monadic second-order logic) that regular trips can be computed by tree-walking automata *with regular site tests*, i.e., tw automata that, additionally, have the ability to test whether the current node belongs to a given regular site (for a fixed, but arbitrary number of regular sites). Clearly, a tw marble/pebble automaton can test whether (t, u) is in site S by dropping its pebble on u, checking (according to Proposition 4.2) whether $\mathrm{mark}(t, u)$ is in the regular tree language $\mathrm{mark}(S)$, with the pebble treated as the mark $(1, 1)$, returning to u, and lifting the pebble. We note that the tw automata with regular site tests are called tw automata *with MSO tests* in [BloEng1, BloEng2].

For the reader who is not familiar with monadic second-order logic, we give a second, direct proof of this lemma (essentially the same as the one in Theorem 8 of [BloEng1] and Theorem 13 of [BloEng2]). Let M be a bottom-up finite tree automaton that recognizes $\mathrm{mark}(T)$. As in the proof of Lemma 3.2 we first describe a nondeterministic automaton A that computes T. The automaton A is started at node u of tree t, and it has to walk to node v, with $(t, u, v) \in T$. Note that we cannot use the same method as in the proof of Lemma 3.2; in fact, we cannot pick up the pebble from u during the simulation of M (during a depth-first traversal of the tree), because there will in general be marbles on the tree at that moment. Thus, a more clever simulation is needed. A walks straight from u to v, along the shortest path in t. At each node on that path it uses its pebble and marbles to compute the relevant states of M. First, A guesses whether v is a descendant of u, an ancestor of u, or neither of the two. Suppose that v is neither a descendant nor an ancestor of u. Then A walks up to the least common ancestor z of u and v (which it has to guess) and walks down to v, guessing its way down. On the way up it computes $\mathrm{state}_{M,t'}(x)$ for every node x between u and z, where $t' = \mathrm{mark}(t, u, v)$, and on the way down it computes $\mathrm{succ}_{M,t'}(y)$ for every node y between z and v; finally it computes $\mathrm{state}_{M,t'}(v)$ and checks whether $\mathrm{state}_{M,t'}(v) \in \mathrm{succ}_{M,t'}(v)$. Let us see in more detail how A can do this. It starts by dropping the pebble on u and computing $\mathrm{state}_{M,t'}(u)$, using the procedure in the proof of Proposition 4.2 and treating the label σ of u as $(\sigma, 1, 0)$. It then lifts the pebble from u, moves one node up, say to x, drops its pebble on x, computes $\mathrm{state}_{M,t'}(u')$ for all children u' of x different from u (and note that this equals $\mathrm{state}_{M,t}(u')$), and applies the state transition function of M to obtain $\mathrm{state}_{M,t'}(x)$. This step is repeated until A arrives at a child, say x_0, of the least common ancestor z. A then computes $\mathrm{succ}_{M,t'}(z)$ (which equals $\mathrm{succ}_{M,t}(z)$) by dropping its pebble on z and, for every

state q of M, simulating M on t under the assumption that M is in state q at node z (and, of course, lifting the pebble from z after doing this). Let y_0 be the child of z on the path from z to v. Now A computes $\text{state}_{M,t}(w)$ for all children w of z different from x_0 and y_0, and uses these states, together with $\text{state}_{M,t'}(x_0)$ and $\text{succ}_{M,t}(z)$, to compute $\text{succ}_{M,t'}(y_0)$ as indicated just before Theorem 4.4. Let y be the child of y_0 on the path from y_0 to v. A then computes $\text{state}_{M,t}(y')$ for all children y' of y_0 different from y, and uses them, together with $\text{succ}_{M,t'}(y_0)$, to compute $\text{succ}_{M,t'}(y)$. This step is now repeated until A arrives in v. Finally A computes $\text{state}_{M,t'}(v)$, treating the label σ of v as $(\sigma, 0, 1)$, and checks whether that state is in $\text{succ}_{M,t'}(v)$.

The cases that v is a descendant or ancestor of u are similar (A just walks down, or just walks up, respectively). They are therefore left to the reader.

It remains to show that A can be made deterministic if T is a functional trip (cf. Theorem 9 of [BloEng1] and Theorem 14 of [BloEng2]). Note that since the procedure in the proof of Proposition 4.2 is deterministic, the automaton A only makes nondeterministic moves when there are no marbles or pebble on the tree. Thus, it suffices to show that for such an automaton a deterministic tw marble/pebble automaton A' can be constructed that computes the same trip. Suppose that A is at node w of t in state q, and that A has several possible moves $m_1, \ldots, m_k$, of which, of course, at most one is successful, i.e., leads to a final state of A (at the destination v). We claim that A' can find out, for each of these moves m, whether m is successful or not. Consider the site $S_{q,m}$ that consists of all (t, w) such that A has a successful computation on t, starting at node w in state q with move m. Since there is a tw marble/pebble automaton that walks to w and simulates A, starting with move m, it follows from Theorem 4.4 that $S_{q,m}$ is a regular site. Thus, as explained in the first paragraph of this proof, A' can test whether (t, w) is in $S_{q,m}$, using the (deterministic) procedure in the proof of Proposition 4.2. □

Taking the last two lemmas together we can state the main result of this paper: the tw marble/pebble automaton is the implementation model of regular trips on trees.

Theorem 4.7 *A trip on trees is regular iff it can be computed by a tw marble/pebble automaton. A functional trip on trees is regular iff it can be computed by a deterministic tw marble/pebble automaton.*

As in the case of strings, since a trip is regular iff it can be expressed in monadic second-order logic, this theorem can be viewed as the generalization of the classical result of Doner and Thatcher/Wright [Don, ThaWri] from tree languages to trips on trees.

As mentioned in the definition of tw marble/pebble automaton, there is a less severe restriction on the behaviour of the automaton that still serves our purposes. To understand this new restriction, we first note that it can always be assumed that there is at most one marble on each node (just take the sets of marble colours as new colours and simulate a set of marbles by one marble). It is easy to see

that, under this assumption, the life times of the marbles are *nested*, i.e., included in one another or disjoint from one another; this is due to the fact that a marble closes off the context. Now, in our new definition of tw marble/pebble automaton, rather than requiring that the pebble can only be dropped or lifted when there are no marbles on the tree, we require that the life times of the marbles and the pebble are nested (see [GloHar] for a similar nesting requirement). Intuitively it means that when the pebble is lifted, the "marble configuration" on the tree has to be exactly the same as when it was dropped (and the involved marbles have not been touched in the mean time). It is shown in Theorem 20 of [vBest] that Theorem 4.4 still holds for these more powerful tw marble/pebble automata, and so does Theorem 4.7. We note that, under this nesting restriction, the restriction that marbles close off the context cannot be dropped.

Example 4.8 *The non-regular monadic tree language* $\{a^n c b^n e \mid n \geq 0\}$ *of Example 4.3 can be recognized as follows, using marbles only, with nested life times. Put a red marble at the root, and a blue marble at the lowest b; then repeat the following step: put a red marble just below the lowest red marble, and put a blue marble just above the highest blue marble. Do this until both the red and blue marble are neighbours of the c-labeled node. Then remove all marbles in the reverse order as they were laid down (i.e., repeatedly the highest blue marble and the lowest red marble).* □

We end this paper by showing that Theorem 4.7 does not hold for tw marble automata, i.e., the pebble is really necessary. Note that it is an open problem whether the marbles are necessary, cf. the Conjecture in the beginning of this section. The proof of the pebble necessity is similar to the one in [Blo, BloEng2] (see Theorem 15 of [BloEng2]).

Theorem 4.9 *There is a (functional) regular trip on trees that cannot be computed by any (nondeterministic) tw marble automaton.*

Proof. Consider the (monadic) ranked alphabet Σ with symbols b and r of rank 1, standing for "black" and "red", respectively, and one symbol e of rank 0. Let T be the trip consisting of all (t, u, v) such that either t has a red root and v is the root, or t has a black root and v is the child of u (viewing the root as the child of the leaf). Thus, either all trips are to the red root, or everybody visits its child. It should be clear that T is regular. Let us now assume that there is a tw marble automaton A that computes T, and derive a contradiction. The idea is that when A starts at any node u of a tree with a black root, it first has to visit the root to be sure that it is not red. Since there is no way for A to remember its starting point u, A cannot anymore find the child of u. Note that when A is at the root, there are no marbles on the tree, except on the root itself.

Formally, consider the tree $t = b^n e$ with $n > s \cdot 2^c$, where s is the number of states of A and c the number of marble colours. Let $t' = rb^{n-1}e$. Thus, t' is t with its root coloured red. Consider, for every node u of t, the successful walk of A from u to its child. Clearly, during this computation A must visit the root, because

otherwise A could make the same computation on t'. As observed above, when A is at the root, all marbles are at the root. Let, at that moment, q_u be the state of A and let M_u be the set of marble(colour)s on the root. Thus, q_u and M_u determine the configuration of A. Hence, by the choice of n, there must be two different nodes u and u' such that, in the corresponding computations, $q_u = q_{u'}$ and $M_u = M_{u'}$ and hence A visits the root in the same configuration in both computations. This implies, however, that A can walk from u to the child of u', a contradiction. □

One may argue that the tw marble/pebble automaton is not a very natural type of automaton, with its rather artificial restrictions on the use of marbles and pebble. The reader is invited to search for a more natural automaton; bread crumbs might be an alternative to marbles and pebbles.

References

[AhoUll] A.V. Aho and J.D. Ullman; Translations on a context free grammar, Inform. and Control 19 (1971), 439–475

[Bir] J.-C. Birget; Two-way automata and length-preserving homomorphisms, Math. Systems Theory 29 (1996), 191–226

[Blo] R. Bloem; *Attribute Grammars and Monadic Second Order Logic*, Master's Thesis, Leiden University, June 1996
`http://www.wi.LeidenUniv.nl/MScThesis/IR96-15.html`

[BloEng1] R. Bloem, J. Engelfriet; Monadic second order logic and node relations on graphs and trees, in Structures in Logic and Computer Science (J.Mycielski, G.Rozenberg, A.Salomaa, eds.), Lecture Notes in Computer Science 1261, Springer-Verlag, 1997, pp.144-161

[BloEng2] R. Bloem, J. Engelfriet; Characterization of properties and relations defined in monadic second order logic on the nodes of trees, Tech. Report 97–03, Leiden University, August 1997
`http://www.wi.LeidenUniv.nl/TechRep/1997/tr97-03.html`

[BluHew] M. Blum and C. Hewitt; Automata on a 2-dimensional tape, in Proc. 8th IEEE Symp. on Switching and Automata Theory, pp.155–160, 1967.

[Büc] J. Büchi; Weak second-order arithmetic and finite automata, Z. Math. Logik Grundlag. Math. 6 (1960), 66–92

[Don] J. Doner; Tree acceptors and some of their applications, J. of Comp. Syst. Sci. 4 (1970), 406–451

[Elg] C. C. Elgot; Decision problems of finite automata and related arithmetics, Trans. Amer. Math. Soc. 98 (1961), 21–51

[ERS] J. Engelfriet, G. Rozenberg, G. Slutzki; Tree transducers, L systems, and two-way machines, J. of Comp. Syst. Sci. 20 (1980), 150-202

[GécSte1] F. Gécseg, M. Steinby; *Tree automata*, Akadémiai Kiadó, Budapest, 1984

[GécSte2] F. Gécseg, M. Steinby; Tree Languages, in G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, Chapter 1, Springer-Verlag, 1997

[GloHar] N. Globerman, D. Harel; Complexity results for two-way and multi-pebble automata and their logics, Theor. Comput. Sci. 169 (1996), 161–184

[HopUll] J.E. Hopcroft, J.D.Ullman; *Introduction to Automata Theory, Languages and Computation*, Addison–Wesley, Reading, Massachusetts, 1979.

[KamSlu] T. Kamimura, G. Slutzki; Parallel and two-way automata on directed ordered acyclic graphs, Inf. and Control 49 (1981), 10–51

[KlaSch] N. Klarlund, M. L. Schwartzbach; Graph Types, in Proc. of the 20th Conference on Principles of Programming Languages, 1993, 196–205

[RabSco] M.O. Rabin, D. Scott; Finite automata and their decision problems, IBM J. Res. Devel. 3 (1959), 115–125

[She] J.C. Shepherdson; The reduction of two-way automata to one-way automata, IBM J. Res. Devel. 3 (1959), 198–200

[ThaWri] J. W. Thatcher, J. B. Wright; Generalized finite automata theory with an application to a decision problem of second-order logic, Math. Systems Theory 2 (1968), 57–81

[vBest] J.P. van Best; *Tree–Walking Automata and Monadic Second Order Logic*, Master's Thesis, Leiden University, July 1998
`http://www.wi.LeidenUniv.nl/MScThesis/IR98-06.html`

Acta Cybernetica **14** (1999) 65-82.

Axiomatizing iteration categories

Z. Ésik*†

Dedicated to Ferenc Gécseg on his 60th birthday

Abstract

We associate an identity with every finite automaton and show that a set of equations consiting of some classical identities as well as the equations associated with a subclass of finite automata is complete for iteration theories if and only if every finite simple group divides the semigroup of an automaton in the given subclass. By taking a special subclass with this property, we arrive at the final result of the paper.

1 Introduction

It has been shown in [3] that the axioms of iteration theories capture the equational properties of the fixed point operation in computer science. For a recent overwiew see also [5]. The first axiomatization of iteration theories was given in [8]. This system was simplified in [9] by proving that some classical identities in conjunction with an identity associated with each finite (simple) group is complete. This result confirms a conjecture in [6] in a general setting. In the present paper we give a further simplification of the iteration theory axioms. We associate an identity with every finite automaton and show that a set of equations consiting of some classical identities as well as the equations associated with a subclass of finite automata is complete if and only if every finite simple group divides the semigroup of an automaton in the given subclass. By taking a special subclass with this property, we arrive at our final result.

In this paper, we define theories in a slightly more general way, so that in this context, we prefer the term iteration categories to iteration theories.

*Supported in part by grant no. T22423 of the National Science Foundation of Hungary, the US-Hungarian Joint Fund under grant no. 351, and by the Austrian-Hungarian Action Foundation.

†Department of Computer Science A. József University Árpád tér 2. 6720 Szeged Hungary

2 Preliminaries

2.1 Conway categories and iteration categories

In any category $\mathcal{C}$, we denote composition by $\cdot$. The identity morphism corresponding to a $\mathcal{C}$-object A will be written id_A, or just id.

We will consider **cartesian categories $\mathcal{C}$ with explicit products**. Thus we assume that for any finite family of $\mathcal{C}$-objects C_i, $i \in [n] = \{1, \ldots, n\}$ we are given a product diagram

$$\pi_j^{C_1 \times \ldots \times C_n} : C_1 \times \ldots \times C_n \quad \to \quad C_j, \quad j \in [n]$$

with the usual universal property. When $f_i : A \to C_i$, $i \in [n]$ is a family of morphisms, the unique mediating morphism $A \to C_1 \times \ldots \times C_n$ will be denoted $\langle f_1, \ldots, f_n \rangle$. This morphism is called the **tupling** of the f_i. In particular, when $n = 0$, the empty tuple is the unique morphism $!_A : A \to 1$, where 1 is the specified terminal object.

We will assume that product is associative on the nose so that $A \times (B \times C) = (A \times B) \times C$, for all objects A, B, C, and diagrams such as

$$\begin{array}{ccc} A \times (B \times C) & \xrightarrow{\pi_2^{A \times (B \times C)}} & B \times C \\ \Big\downarrow \mathrm{id} & & \Big\downarrow \pi_2^{B \times C} \\ (A \times B) \times C & \xrightarrow[\pi_2^{(A \times B) \times C}]{} & C \end{array}$$

commute. In particular, we assume that for each object A the projection morphism $\pi_1^A : A \to A$ is the identity morphism id_A. It follows that $\langle f \rangle = f$ for all $f : A \to B$. We also assume that

$$\langle f, ! \rangle = \langle !, f \rangle = f,$$

for all morphisms $f : A \to B$.

In the sequel we will call tuplings of projections as **base morphism**. Note that any base morphism $A^n \to A^m$ corresponds to a function $\rho : [m] \to [n]$. In fact the base morphism $A^n \to A^m$ determined by ρ is given by

$$\langle \pi_{1\rho}^{A^n}, \ldots, \pi_{m\rho}^{A^n} \rangle.$$

We will call a base morphism corresponding to a permutation $[n] \to [n]$ a **base permutation**.

For any cartesian category $\mathcal{C}$ we define the bifunctor $\mathcal{C} \times \mathcal{C} \to \mathcal{C}$ by

$$f \times g \quad = \quad \langle f \cdot \pi_1^{C \times D}, \; g \cdot \pi_2^{C \times D} \rangle,$$

for all $f : C \to A$, $g : D \to B$.

DEFINITION 2.1 *A* **preiteration category** *is a cartesian category C equipped with an* **external dagger operation**

$$^\dagger : C(A \times B, A) \quad \to \quad C(B, A),$$

see [4].

The **Conway identities** are the **parameter** (1), **double dagger** (2) and **composition identities** (3) given below.

$$(f \cdot (\mathrm{id}_A \times g))^\dagger \quad = \quad f^\dagger \cdot g, \tag{1}$$

all $f : A \times B \to A$, $g : C \to B$,

$$f^{\dagger\dagger} \quad = \quad (f \cdot (\Delta \times \mathrm{id}_C))^\dagger, \tag{2}$$

where $f : A \times A \times C \to A$ and where Δ is the diagonal morphism $\langle \mathrm{id}_A, \mathrm{id}_A \rangle : A \to A \times A$.

$$(f \cdot \langle g, \pi_2^{A \times C} \rangle)^\dagger \quad = \quad f \cdot \langle (g \cdot \langle f, \pi_2^{B \times C} \rangle)^\dagger, \ \pi_2^{B \times C} \rangle, \tag{3}$$

for all $f : B \times C \to A$, $g : A \times C \to B$. Note that the **fixed point identity**

$$f^\dagger \quad = \quad f \cdot \langle f^\dagger, \mathrm{id}_C \rangle, \quad f : A \times C \to A$$

is a particular subcase of the composition identity.

DEFINITION 2.2 [3] *A* **Conway category** *is a preiteration category satisfying the Conway identities.*

Conway categories satisfy several other non-trivial identities including the **Bekič identity** [1] (called the **pairing identity** in [3]):

$$\langle f, g \rangle^\dagger \quad = \quad \langle f^\dagger \cdot \langle h^\dagger, \mathrm{id}_C \rangle, \ h \rangle^\dagger,$$

for all $f : A \times B \times C \to A$ and $g : A \times B \times C \to B$, where

$$h \quad = \quad g \cdot \langle f^\dagger, \mathrm{id}_{B \times C} \rangle : B \times C \to B.$$

We will also make use of the **permutation identity**

$$(\pi \cdot f \cdot (\pi^{-1} \times \mathrm{id}_C))^\dagger \quad = \quad \pi \cdot f^\dagger,$$

for all $f : A^n \times C \to A^n$ and all base permutations $\pi : A^n \to A^n$. Another useful identity is given by the next lemma.

LEMMA 2.3 *In any Conway category C,*

$$f^{\dagger \cdots \dagger} \quad = \quad (f \cdot (\Delta_n \times \mathrm{id}_p))^\dagger,$$

for all morphisms $f : A^n \times C \to A$, where there are $n > 1$ consecutive daggers on the left hand side and where Δ_n is the diagonal morphism $\langle \mathrm{id}_A, \ldots, \mathrm{id}_A \rangle : A \to A^n$.

A full description of the valid identities of Conway categories is given in [2], where it is proved that the problem of deciding whether an equation holds in all Conway categories is PSPACE-complete. It is shown in [4] that the parameter identity corresponds to a naturality condition and that the double dagger identity to a dinaturality condition of the dagger operation.

As argued in [3], all of the fixed point models in computer science satisfy at least the Conway identities. For example, for any set S, the category $\mathbf{Cpo}^S$ of S-sorted cpo's and continuous functions satisfies the Conway identities. In this category, there is a cpo A_w corresponding to any word $w \in S^*$. When $w = s_1 \dots s_n$, the cpo A_w is determined by the cpo's A_{s_i}, in fact A_w is the product $A_{s_1} \times \dots \times A_{s_n}$. The morphisms $A_w \to A_v$ are the continuous (or order preserving) functions $A_w \to A_v$, and the dagger operation is defined by least fixed points.

We give a semantic definition of iteration categories. For a syntactic characterization the reader is referred to Section 3.

DEFINITION 2.4 *An* **iteration category** *is a preiteration category equipped with a dagger operation which satisfies all of the identities that hold in the categories* $\mathbf{Cpo}^S$.

It is shown in [3], see also [5], that the iteration category identities are *the* standard identities of the fixed point operation in computer science.

2.2 Automata and semigroups

Except for free semigroups, all semigroups will be assumed to be finite. The product of the elements s, t of a semigroup S will be written $s \circ t$, or just st. A subgroup of a semigroup S is a subsemigroup of S which is a group. Following [7, 12], we say that a semigroup S **divides** a semigroup S', denoted $S|S'$, if S is a homomorphic image of a subsemigroup of S'. It is known that the division relation is transitive (and reflexive). Further, a group G divides a semigroup S if and only if G is a homomorphic image of a subgroup of S. A group G is called **simple** if it is nontrivial and has no proper nontrivial normal subgroup.

Suppose that X is a finite nonempty set. An X-**automaton** $\mathbf{Q} = (Q, X, \circ)$ is a finite nonempty set Q equipped with a (right) action of X on Q:

$$\begin{array}{rcl} \circ : Q \times X & \to & Q \\ (q, x) & \mapsto & q \circ x. \end{array}$$

We will usually write qx for $q \circ x$ and (Q, X) for $(Q, X, \circ)$. The action of X on Q may be extended to an action of the free semigroup X^+ of all finite nonempty words over X such that

$$q(ux) = (qu)x$$

for all $q \in Q$, $u \in X^+$ and $x \in X$.

Suppose that $\mathbf{Q} = (Q, X)$ is an automaton. A letter $x \in X$ is a permutation letter (reset letter, respectively) if the function

$$q \mapsto qx, \quad q \in Q$$

induced by x is a permutation (constant map, respectively) on Q. We call $\mathbf{Q}$ a **permutation automaton** (**reset automaton**, respectively) if each letter $x \in X$ is a permutation letter (reset letter, respectively). Further, we call $\mathbf{Q}$ a **permutation–reset automaton** if each $x \in X$ is either a permutation letter or a reset letter. For example, the automaton $\mathbf{U} = (\{q_1, q_2\}, \{x_1, x_2, x_3\})$ equipped with the action

$$\begin{aligned} q_i x_j &= q_j \\ q_i x_3 &= q_i, \quad i, j \in [2], \end{aligned}$$

is a permutation-reset automaton, called the **two-state identity-reset automaton**. This automaton is important in the Krohn-Rhodes decomposition theorem, see [11]. In our arguments we will also make use of counters. A **counter of length** n is a (permutation) automaton $(Q, \{x\})$ such that $Q = \{q_0, \ldots, q_{n-1}\}$ has n elements and letter x induces the cyclic permutation $q_i \mapsto q_{i+1 \bmod n}$.

Homomorphisms, subautomata and **congruences** of automata are defined in the usual way. The automaton (Q, X) is called a **renaming** of the automaton (Q, Y) if there is a function $\varphi : X \to Y$ such that

$$qx = q(x\varphi),$$

for all $q \in Q$ and $x \in X$.

Suppose that $\mathbf{Q} = (Q, X)$ is an automaton. Recall that each word $u \in X^+$ induces a function $Q \to Q$. Equipped with the operation of composition that we now write in diagrammatic order, these functions form a semigroup denoted $S(\mathbf{Q})$. We call $S(\mathbf{Q})$ the **semigroup of** $\mathbf{Q}$. For example, the semigroup of a counter of length n is a cyclic group of order n. When $\mathbf{Q}$ is a permutation automaton, each element of $S(\mathbf{Q})$ is a permutation of the set Q, so that $S(\mathbf{Q})$ is a group. An automaton $\mathbf{Q}$ is called **aperiodic** [7], if each subgroup of $S(\mathbf{Q})$ is trivial. For example, each reset automaton, or more generally, each **definite** automaton [7] is aperiodic. The automaton $\mathbf{U}$ is also aperiodic. We will denote the class of aperiodic automata by $\mathcal{AP}$.

The concept of aperiodic automata may be generalized. Suppose that $\mathcal{G}$ is a class of simple groups closed under division. *We let $\mathcal{Q}_\mathcal{G}$ denote the class of all automata $\mathbf{Q}$ such that any simple group divisor of $S(\mathbf{Q})$ is in $\mathcal{G}$.* Thus, when $\mathcal{G}$ is empty, $\mathcal{Q}_\mathcal{G}$ is the class $\mathcal{AP}$. When $\mathcal{G}$ is the class of all cyclic groups of prime order, $\mathcal{Q}_\mathcal{G}$ is known as the class of **solvable automata**. We denote this class by $\mathcal{SOL}$. We will also make use of the following notation. Suppose that $m \geq 1$ is an integer. Then we let $\mathcal{SOL}_m$ denote the class of all (solvable) automata $\mathbf{Q}$ such that any simple group divisor of $S(\mathbf{Q})$ is a cyclic group of prime order p which divides m. Thus, $\mathcal{SOL}_m = \mathcal{SOL}_n$ if and only if m and n have the same prime divisors. Note that $\mathcal{SOL}_1 = \mathcal{AP}$.

When (Q, X) is an automaton such that $X = S$ is a semigroup and the action is compatible with the semigroup operation, i.e.,

$$q(st) = (qs)t$$

for all $q \in Q$ and $s, t \in S$, we call the automaton (A, S) a **transformation semigroup**. (Note that we are not requiring that the action is faithful.) When S is a group with unit e and

$$qe = q,$$

for all $q \in Q$, (Q, S) is a **transformation group**. See [7]. Note that each transformation group is a permutation automaton.

For each semigroup S there is a corresponding transformation semigroup (S, S) equipped with the natural self action $(s, t) \mapsto st$. When S is a group, (S, S) is a transformation group.

Following [11], we now define cascade compositions (or α_0-products) of automata. For this reason, suppose that $\mathbf{Q}_i = (Q_i, X_i)$, $i \in [n]$, $n > 0$, are given automata. Moreover, suppose that X is a new finite nonempty set and for each $i \in [n]$ we are given a function

$$\varphi_i : Q_1 \times \ldots \times Q_{i-1} \times X \to X_i.$$

Then the **cascade composition**

$$\prod_{i \in [n]} \mathbf{Q}_i[X, \varphi_i]$$

determined by the functions φ_i is the automaton $(\prod_{i \in [n]} Q_i, X)$ equipped with the X-action

$$(q_1, \ldots, q_n)x = (q_1 y_1, \ldots, q_n y_n),$$

where $y_i = \varphi_i(q_1, \ldots, q_{i-1}, x)$, for all i. Note that when $n = 1$, a cascade composition is just a renaming of $\mathbf{Q}_1$. We will sometimes denote the above cascade composition as

$$\mathbf{Q}_1 \times \ldots \times \mathbf{Q}_n[X, \varphi_1, \ldots, \varphi_n].$$

Two particular subcases of the cascade composition are also important, the quasi-direct product and the direct product. We call the above cascade composition a **quasi-direct product** if each function φ_i is independent of its first $i - 1$ arguments, so that each φ_i can be considered as a function $X \to X_i$. If for each i also $X = X_i$ and φ_i is the identity function $X \to X$, then the quasi-direct product is the **direct product** $\prod_{i \in [n]} \mathbf{Q}_i$.

We will say that an automaton (Q, X) **has an identity letter** if some $x \in X$ induces the identity function $Q \to Q$. Given $\mathbf{Q}$, we will denote by $\mathbf{Q}^1$ an automaton obtained from $\mathbf{Q}$ by adding a letter inducing the identity function $Q \to Q$, if $\mathbf{Q}$ has no such letter. Otherwise $\mathbf{Q}^1$ is just $\mathbf{Q}$. This notation is extended to classes of automata in a natural way.

3 Review

In this section we review some of the results of [9] and [10].

Suppose that $\mathbf{Q} = (Q, X)$ is a finite automaton such that $Q = [n]$ and $X = [m]$, for some integers n and m. For each preiteration category $\mathcal{C}$ and object A in $\mathcal{C}$, we associate with $\mathbf{Q}$ the base morphisms $\rho_q^{\mathbf{Q}} : A^n \to A^m$, $q \in Q$. For each q, $\rho_q^{\mathbf{Q}}$ corresponds to the map

$$\begin{array}{ccc} [m] & \to & [n] \\ x & \mapsto & qx. \end{array}$$

Thus,

$$\rho_q^{\mathbf{Q}} = \langle \pi_{q1}^{A^n}, \ldots, \pi_{qm}^{A^n} \rangle.$$

(Recall that $X = [m]$, so that for each $q \in Q = [n]$ and $i \in [m]$, qi is a state of the automaton $\mathbf{Q}$.) The morphisms $\rho_q^{\mathbf{Q}}$, denoted sometimes just ρ_q, are called the **base morphisms associated with the automaton $\mathbf{Q}$.**

We define, for each $g : A^m \times C \to A$,

$$g_{\mathbf{Q}} = \langle g \cdot (\rho_1 \times \mathrm{id}_C), \ldots, g \cdot (\rho_n \times \mathrm{id}_C) \rangle : A^n \times C \to A^n.$$

DEFINITION 3.1 *The* **automaton-identity $\Gamma(\mathbf{Q})$** *associated with* $\mathbf{Q}$ *is the equation*

$$(g_{\mathbf{Q}})^\dagger = \Delta_n \cdot (g \cdot (\Delta_m \times \mathrm{id}_C))^\dagger, \quad g : A^m \times C \to A. \tag{4}$$

In preiteration categories satisfying the permutation identity we can associate an equation with any automaton not just with those defined on sets of the form $[m]$. In such categories, equations associated with isomorphic automata are equivalent.

Since any transformation semigroup is an automaton, the above definition associates an identity $\Gamma(Q, S)$ with each transformation semigroup (Q, S). When (Q, S) is the transformation semigroup (S, S) equipped with the natural self action, we denote $\Gamma(S, S)$ by $\Gamma(S)$ and call this identity the **semigroup-identity associated with** S. When S is group, $\Gamma(S)$ is a **group-identity**.

The above notation may be extended to classes of automata and semigroups. When $\mathcal{Q}$ is a class of finite automata, $\Gamma(\mathcal{Q})$ consists of all identities $\Gamma(\mathbf{Q})$, $\mathbf{Q} \in \mathcal{Q}$. When $\mathcal{S}$ is a class of finite semigroups, $\Gamma(\mathcal{S})$ is defined similarly.

The axiomatization of iteration categories given in the next theorem is a reformulation of the main result of [8].

THEOREM 3.2 *A Conway category $\mathcal{C}$ is an iteration category if and only if each automaton identity holds in $\mathcal{C}$.*

The following stronger results were proved in [9] and [10].

THEOREM 3.3 *Suppose that $\mathcal{S}$ is a given class of semigroups and $\mathbf{Q}$ is an automaton. Then the automaton identity $\Gamma(\mathbf{Q})$ associated with $\mathbf{Q}$ holds in all Conway categories satisfying the semigroup-identities $\Gamma(\mathcal{S})$ if and only if every simple group divisor of $S(\mathbf{Q})$ divides one of the semigroups in $\mathcal{S}$.*

In particular, an automaton identity $\Gamma(\mathbf{Q})$ holds in all Conway categories if and only if $\mathbf{Q} \in \mathcal{AP}$. And if $\mathcal{G}$ is any class of simple groups closed under division, then $\Gamma(\mathbf{Q})$ holds in all Conway categories satisfying the group-identities $\Gamma(\mathcal{G})$ if and only if $\mathbf{Q} \in \mathcal{Q}_{\mathcal{G}}$.

COROLLARY 3.4 [9] *A Conway category is an iteration category if and only if it satisfies the group-identities. Given a class $\mathcal{S}$ of finite semigroups, consider the set of equations $\Gamma(S)$ associated with the semigroups in $\mathcal{S}$. The system consisting of the Conway identities and the equations $\Gamma(S)$ is complete for iteration categories if and only if for every simple group G there is a semigroup $S \in \mathcal{S}$ such that $G|S$.*

In the course of proving Theorem 3.3, the following facts were established in [9].

LEMMA 3.5 *Suppose that $\mathbf{Q}$ is a subautomaton or a renaming of $\mathbf{Q}'$. If C is a Conway category with $C \models \Gamma(\mathbf{Q}')$ then $C \models \Gamma(\mathbf{Q})$.*

LEMMA 3.6 *Let C be a Conway category and suppose that $\mathbf{Q} = \prod_{i\in[n]} \mathbf{Q}_i[X, \varphi_i]$ is a cascade composition. If $C \models \Gamma(\mathbf{Q}_i)$ for all $i \in [n]$, then $C \models \Gamma(\mathbf{Q})$. Moreover, if φ_1 is surjective and if $C \models \Gamma(\mathbf{Q})$ and $C \models \Gamma(\mathbf{Q}_i)$ for all $i > 1$, then $C \models \Gamma(\mathbf{Q}_1)$.*

4 Main results

The main results of this paper are Theorem 4.2, Corollary 4.4 and Theorem 4.5 below. In order to formulate these results, we need one more definition.

The **powers** $f^k : A \times C \to A$, $k \geq 0$, of a morphism $f : A \times C \to A$ in a cartesian category are defined by induction:

$$\begin{aligned} f^0 &= \pi_1^{A\times C} \\ f^{k+1} &= f \cdot \langle f^k, \pi_2^{A\times C} \rangle. \end{aligned}$$

DEFINITION 4.1 *For each $m \geq 1$, the mth* **power identity** *is the equation $\mathbf{P}_m$*

$$(f^m)^\dagger = f^\dagger, \quad f : A \times C \to A.$$

Note that this identity is nontrivial only if $m > 1$. We will prove

THEOREM 4.2 *Suppose that $\mathcal{Q}$ is a class of automata and $\mathbf{Q}$ is an automaton such that every simple group divisor of $S(\mathbf{Q})$ divides the semigroup of some automaton in $\mathcal{Q}$. If C is a Conway category satisfying the identities $\Gamma(\mathcal{Q})$ and a nontrivial power identity, then $C \models \Gamma(\mathbf{Q})$.*

COROLLARY 4.3 *Suppose that a renaming of some automaton in $\mathcal{Q}$ contains a nontrivial counter as a subautomaton. Then the identity $\Gamma(\mathbf{Q})$ associated with an automaton $\mathbf{Q}$ holds in all Conway categories satsifying the identities $\Gamma(\mathcal{Q})$ if and only if every simple group divisor of $S(\mathbf{Q})$ divides the semigroup of an automaton in $\mathcal{Q}$.*

From Corollary 4.3 and Theorem 3.2 we immediately have

COROLLARY 4.4 *Suppose that a renaming of an automaton in $\mathcal{Q}$ contains a nontrivial counter. If every (simple) group is a divisor of the semigroup of an automaton in $\mathcal{Q}$, then the Conway identities and the automaton identites in $\mathbf{S}(\mathcal{Q})$ are complete for iteration categories.*

Conversely, if $\mathcal{Q}$ is any class of finite automata such that the Conway identities, the power identities, and the automaton identities in $\Gamma(\mathcal{Q})$ are complete for iteration categories, then every (simple) group divides the semigroup of an automaton in $\mathcal{Q}$.

Let us now define, for each $n \geq 3$, the identity $\mathbf{S}_n$

$$(f \cdot (\Delta_2 \times \mathrm{id}_C) \cdot \langle f \cdot \langle \pi_1^{A\times C},\ (f^\dagger)^{n-2}, \pi_2^{A\times C}\rangle,\ \pi_2^{A\times C}\rangle)^\dagger \quad = \quad (f \cdot (\Delta_2 \times \mathrm{id}_C))^\dagger,$$

where f is any morphism $A^2 \times C \longrightarrow A$ in a preiteration category. This identity is a generalization of an identity of regular sets introduced by John Conway in [6]. As an application of Theorem 4.2, we will prove

THEOREM 4.5 *The Conway identities and the equations $\mathbf{S}_n$, for all $n \geq 3$, are complete for iteration categories.*

In order to establish these results, we need to derive the identity $\Gamma(G)$ associated with a group G dividing the semigroup of an automaton $\mathbf{Q}$ from the the identity $\Gamma(\mathbf{Q})$, a nontrivial power identity, and the Conway identities.

5 Identities associated with solvable automata

In this section, we show that in Conway categories, the mth power identity is equivalent to the identity associated with a counter of length m. We then proceed to prove that an automaton identity $\Gamma(\mathbf{Q})$ holds in all Conway categories satisfying the mth power identity if and only if $\mathbf{Q} \in \mathcal{SOL}_m$. We start with a technical lemma.

LEMMA 5.1 *Suppose that C is a Conway category satisfying the identity $\Gamma(\mathbf{Q})$ associated with a finite automaton $\mathbf{Q}$. Then $C \models \Gamma(\mathbf{Q}^1)$.*

Proof. Suppose that $\mathbf{Q} = (Q, X)$. If $\mathbf{Q}$ has a letter inducing the identity function $Q \longrightarrow Q$ then $\mathbf{Q}^1 = \mathbf{Q}$ and there is nothing to prove. Otherwise $\mathbf{Q}^1 = (Q, Y)$ with $Y = \{y\} \cup X$ such that y induces the identity function $Q \longrightarrow Q$ and each $x \in X$ induces the same function in $\mathbf{Q}$ as in $\mathbf{Q}^1$. In our argument, we assume that $Q = [n]$, $X = \{i : 2 \leq i \leq m+1\}$, so that $Y = [m+1]$ and $y = 1$.

Suppose that C is a Conway category and A and C are objects in C. Define

$$\begin{aligned} \rho_i &= \rho_i^{\mathbf{Q}} : A^n \longrightarrow A^m \\ \sigma_i &= \rho_i^{\mathbf{Q}^1} : A^n \longrightarrow A^{1+m}, \end{aligned}$$

for all $i \in [n]$. Then we have

$$\sigma_i \quad = \quad \langle \pi_i^{A^n}, \rho_i \rangle, \tag{5}$$

for all $i \in [n]$. We complete the argument by using the following sublemma whose proof is omitted.

SUBLEMMA 5.2 *Suppose that* $f_i : A^{1+n} \times C \longrightarrow A$, $i \in [n]$ *in a Conway category* $\mathcal{C}$. *Then*

$$\langle f_1 \cdot \langle \pi_1^{A^n \times C}, \mathrm{id}_{A^n \times C} \rangle, \ldots, f_n \cdot \langle \pi_n^{A^n \times C}, \mathrm{id}_{A^n \times C} \rangle \rangle^\dagger \quad = \quad \langle f_1^\dagger, \ldots, f_n^\dagger \rangle^\dagger.$$

Suppose now that $f : A^{1+m} \times C \longrightarrow A$. Then, by Sublemma 5.2, equation (5), and the parameter identity,

$$\begin{aligned} (f_{\mathbf{Q}^1})^\dagger &= \langle f^\dagger \cdot (\rho_1 \times \mathrm{id}_C), \ldots, f^\dagger \cdot (\rho_n \times \mathrm{id}_C) \rangle^\dagger \\ &= (g_{\mathbf{Q}})^\dagger, \end{aligned}$$

where g is the morphism $f^\dagger$. Thus, since $\mathcal{C} \models \Gamma(\mathbf{Q})$, we have

$$\begin{aligned} (f_{\mathbf{Q}^1})^\dagger &= (g_{\mathbf{Q}})^\dagger \\ &= \Delta_n \cdot (f^\dagger \cdot (\Delta_m \times \mathrm{id}_C))^\dagger \\ &= \Delta_n \cdot (f \cdot (\Delta_{m+1} \times \mathrm{id}_C))^\dagger, \end{aligned}$$

where the last step follows from Lemma 2.3. □

The following fact is obvious.

LEMMA 5.3 *Suppose that* $\mathcal{C}$ *is a preiteration category and* $m, n \geq 1$. *If* $\mathcal{C} \models \mathbf{P}_m$ *and* $\mathcal{C} \models \mathbf{P}_n$, *then* $\mathcal{C} \models \mathbf{P}_{mn}$.

For the rest of this section, for each $m \geq 1$ we let $\mathbf{K}_m$ denote a counter of length m.

LEMMA 5.4 *For any Conway category* $\mathcal{C}$ *and* $m \geq 1$, $\mathcal{C} \models \mathbf{P}_m$ *if and only if* $\mathcal{C} \models \Gamma(\mathbf{K}_m)$.

Proof. This is obvious if $m = 1$, so we assume $m > 1$. It is easy to see that $\mathcal{C} \models \Gamma(\mathbf{K}_m)$ if and only if

$$\pi_1^{A^m} \cdot (f_{\mathbf{K}_m})^\dagger \quad = \quad f^\dagger,$$

for all $f : A \times C \longrightarrow A$. But since $\mathcal{C}$ is a Conway category,

$$\pi_1^{A^m} \cdot (f_{\mathbf{K}_m})^\dagger \quad = \quad (f^m)^\dagger.$$

Indeed, we have

$$f_{\mathbf{K}_m} \quad = \quad \langle f \cdot (\pi_2^{A^m} \times \mathrm{id}_C), \ldots, f \cdot (\pi_m^{A^m} \times \mathrm{id}_C),\ f \cdot (\pi_1^{A^m} \times \mathrm{id}_C) \rangle : A^m \times C \longrightarrow A^m.$$

Define

$$g \;=\; \langle f\cdot(\pi_2^{A^m}\times \mathrm{id}_C),\ldots,f\cdot(\pi_m^{A^m}\times \mathrm{id}_C)\rangle : A^m\times C\to A^{m-1}.$$

Then

$$g^{m-1} \;=\; \langle f^{m-1},\ldots,f\rangle\cdot(\pi_m^{A^m}\times \mathrm{id}_C).$$

Thus, by the fixed point identity,

$$\begin{aligned} g^\dagger &= g^{m-1}\cdot\langle g^\dagger,\mathrm{id}_{A\times C}\rangle \\ &= \langle f^{m-1},\ldots,f\rangle : A\times C\to A^{m-1}. \end{aligned}$$

Thus, by the pairing identity,

$$\begin{aligned} \pi_1^{A^m}\cdot(f_{\mathbf{K}_m})^\dagger &= \pi_1^{A^{m-1}}\cdot g^\dagger\cdot\langle h^\dagger,\mathrm{id}_C\rangle \\ &= f^{m-1}\cdot\langle h^\dagger,\mathrm{id}_C\rangle, \end{aligned}$$

where

$$\begin{aligned} h &= f\cdot(\pi_1^{A^m}\times \mathrm{id}_C)\cdot\langle g^\dagger,\mathrm{id}_{A\times C}\rangle \\ &= f\cdot\langle f^{m-1},\pi_2^{A\times C}\rangle \\ &= f^m. \end{aligned}$$

Thus, $h^\dagger=(f^m)^\dagger$ and

$$\begin{aligned} \pi_1^{A^m}\cdot(f_{\mathbf{K}_m})^\dagger &= f^{m-1}\cdot\langle(f^m)^\dagger,\mathrm{id}_C\rangle \\ &= (f^m)^\dagger, \end{aligned}$$

by the composition identity. □

Suppose that $\mathcal{C}$ is a Conway category satisfying the mth power identity $\mathbf{P}_m$. Let Z_m denote the cyclic group Z/mZ of order m. In order to prove that $\mathcal{C}$ satisfies the group-identity $\Gamma(Z_m)$ we need a technical consruction involving automata.

We represent Z_m as the set $\{0,\ldots,m-1\}$ with group operation

$$(i,j) \;\mapsto\; i+j \bmod m.$$

Similarly, we represent $\mathbf{K}_m^1$ as the automaton (Z_m,X), where $X=\{0,1\}$, so that X is a generating set of the group Z_m. The action of X on Z_m is defined by the group operation. Define the quasi-direct product

$$\mathbf{A} \;=\; (A,Z_m) \;=\; (Z_m,Z_m)\times(Z_m,X)^{m-2}[Z_m,\varphi_1,\ldots,\varphi_{m-1}]$$

by

$$\begin{aligned} j\varphi_1 &= j \quad ; \\ j\varphi_i &= \begin{cases} 0 & \text{if } j\neq i \\ 1 & \text{if } j=i, \end{cases} \end{aligned}$$

for all $j \in \{0, \ldots, m-1\}$ and $i \in \{2, \ldots, m-1\}$. Moreover, define

$$\mathbf{B} \quad = \quad (B, Z_m) \quad = \quad (Z_m, X)^{m-1}[Z_m, \psi_1, \ldots, \psi_{m-1}]$$

by

$$j\psi_i \quad = \quad \begin{cases} 0 & \text{if } j \neq i \\ 1 & \text{if } j = i, \end{cases}$$

for all $j \in \{0, \ldots, m-1\}$ and $i \in \{1, \ldots, m-1\}$. Note that $A = B = Z_m^{m-1}$.

LEMMA 5.5 *The automata* $\mathbf{A}$ *and* $\mathbf{B}$ *are isomorphic.*

Proof. Define

$$\begin{aligned} \mu : B \quad &\to \quad A \\ (i_1, \ldots, i_{m-1}) \quad &\mapsto \quad (\sum_{j=1}^{m-1} i_j \cdot j,\ i_2, \ldots, i_{m-1}), \end{aligned}$$

where the sum is taken mod m. Then μ is a bijection. Suppose that $k \in \{0, \ldots, m-1\}$, $k \neq 0$. Then, in $\mathbf{B}$,

$$(i_1, \ldots, i_{m-1}) \circ k \quad = \quad (i_1, \ldots, i_k + 1, \ldots, i_{m-1}).$$

Moreover, in $\mathbf{A}$,

$$\mu(i_1, \ldots, i_{m-1}) \circ k \quad = \quad (k + \sum_{j=1}^{m-1} i_j \cdot j,\ i_2, \ldots, i_k + 1, \ldots, i_{m-1}),$$

if $k > 1$, and

$$\mu(i_1, \ldots, i_{m-1}) \circ k \quad = \quad (k + \sum_{j=1}^{m-1} i_j \cdot j,\ i_2, \ldots, i_{m-1}),$$

if $k = 1$. In either case, μ preserves the action. □

Thus, by Lemmas 5.4, 5.1 and 3.6, if $\mathcal{C}$ is a Conway category satisfying the mth power identity, then, $T \models \Gamma(\mathbf{B})$. But by Lemma 5.5, $\mathbf{A}$ is isomorphic to $\mathbf{B}$, so that $T \models \Gamma(\mathbf{A})$. But then, again by Lemma 3.6, $\mathcal{C} \models \Gamma(Z_m, Z_m)$. We have proved

LEMMA 5.6 *Suppose that* $\mathcal{C}$ *is a Conway category satisfying the* m*th power identity, for some* $m \geq 1$. *Then* $\mathcal{C} \models \Gamma(Z_m)$.

THEOREM 5.7 *Let* $m \geq 1$ *be any fixed integer. The identity* $\Gamma(\mathbf{Q})$ *associated with an automaton* $\mathbf{Q}$ *holds in all Conway categories satisfying the* m*th power identity if and only if* $\mathbf{Q} \in \mathcal{SOL}_m$.

Proof. Suppose that $\mathcal{C}$ is a Conway category with $\mathcal{C} \models \mathbf{P}_m$. Then, by Lemma 5.6 and Theorem 3.3, $\mathcal{C}$ satisfies the identity $\Gamma(\mathbf{Q})$ associated with any automaton $\mathbf{Q} \in \mathcal{SOL}_m$. On the other hand, if $\mathbf{Q} \notin \mathcal{SOL}_m$, then by Theorem 3.3 there is a Conway category $\mathcal{C}_0$ satisfying $\Gamma(Z_m)$ such that $\Gamma(\mathbf{Q})$ does not hold in $\mathcal{C}_0$. But by Lemma 5.4, the mth power identity holds in $\mathcal{C}_0$. □

COROLLARY 5.8 *The identity associated with an automaton* $\mathbf{Q}$ *holds in all Conway categories satisfying all of power identities if and only if* $\mathbf{Q} \in \mathcal{SOL}$.

6 Proof of Theorem 4.2

Suppose that $\mathbf{Q} = (Q, X)$ is an automaton having an identity letter. Recall that X^+ denotes the free semigroup of all nonempty words over X. Below we write X^* for $X^+ \cup \{\lambda\}$, where λ is the empty word.

Let S denote the semigroup $S(\mathbf{Q})$ and let G be a subgroup of S. Since $\mathbf{Q}$ has an identity letter, S is a monoid whose unit is the identity function $Q \to Q$. Moreover, *there is an integer* $k_0 > 0$ *such that for each* $k \geq k_0$, *any function in* S *is induced by a word in* X^+ *of length* k. For the rest of this section, for any integer $n \geq 0$, we denote by X^n the set of all words $u \in X^*$ of length $|u| = n$. Similarly, G^n is the set of all words in G^* of length n.

For a given word $u \in X^+$, we denote by $\overline{u}$ the function $Q \to Q$ induced by u in $\mathbf{Q}$. Also, when $u = g_1 \dots g_n \in G^+$, then we denote by $\overline{u}$ the composite $g_1 \circ \dots \circ g_n$ of the functions $g_1, \dots, g_n$. (Recall that we write composition in S from left to right.) For a state $q \in Q$, we will just write qu for $q\overline{u}$.

Fix an integer $k \geq k_0$. There exists a function $\psi : G^k \to X^k$ such that $\overline{u} = \overline{u\psi}$ for all $u \in G^k$. Given such a function ψ, for every word $u \in G^k$ we define $u\psi_1 = \mathrm{first}_1(u\psi)$ to be the first letter of $u\psi$, and $u\psi_2 = \mathrm{last}_{k-1}(u\psi)$ to be the suffix of length $k-1$ of $u\psi$. Thus, $u\psi = (u\psi_1)(u\psi_2)$.

Let

$$R \;=\; \{(i, u, v, w) : i \in [k],\ u \in G^i,\ v \in X^{k-i},\ w \in G^k,\ v = \mathrm{last}_{k-i}(w\psi)\}.$$

We turn R into a G-automaton by defining

$$(i, u, v, w) \circ g \;=\; \begin{cases} (i+1, ug, v', w) & \text{if } v = xv' \text{ with } x \in X \\ (1, g, u\psi_2, u) & \text{if } v = \lambda. \end{cases}$$

LEMMA 6.1 *The automaton* $\mathbf{R} = (R, S)$ *is isomorphic to a subautomaton of a cascade composition of a counter of length* k *with aperiodic automata.*

Proof. When $k = 1$ the automaton $\mathbf{R}$ is definite and hence our claim is obvious. Thus, in the rest of the argument, we assume that $k > 1$. Let $\mathbf{K}$ denote the counter $([k], \{z\})$ such that z induces the cyclic permutation $(12 \dots k)$. Let $\mathbf{R}_1 = (G^k, G \times [k])$ and $\mathbf{R}_2 = (X^{k-1}, X \cup X^{k-1})$ be equipped with the following actions:

$$g_1 \dots g_k \circ (g, i) \;=\; \begin{cases} g_1 \dots g_{i-1} g g_{i+1} \dots g_k & \text{if } i \neq 1 \\ g g_0^{k-1} & \text{if } i = 1 \end{cases}$$

$$\begin{aligned} x_1 \dots x_{k-1} \circ x &= x_2 \dots x_{k-1} x \\ x_1 \dots x_{k-1} \circ x'_1 \dots x'_{k-1} &= x'_1 \dots x'_{k-1}, \end{aligned}$$

where $i \in [k]$, $g, g_j \in G$, for all $j \in [k]$, and $x, x_j, x'_j \in X$, for all $j \in [k-1]$, and where g_0 denotes a fixed element (say the unit element) of the group G. Moreover, let $\mathbf{R}_3$ be the automaton $(G^k, G^k \cup \{z\})$ with action

$$\begin{aligned} u \circ v &= v \\ u \circ z &= u, \end{aligned}$$

for all $u, v \in G^k$.

Define the cascade composition $\mathbf{R}' = \mathbf{K} \times \mathbf{R}_1 \times \mathbf{R}_2 \times \mathbf{R}_3[G, \varphi_1, \varphi_2, \varphi_3, \varphi_4]$ as follows. For all $i \in [k]$, $u \in G^k$, $v \in X^{k-1}$ and $g \in G$,

$$\begin{aligned} \varphi_1(g) &= z \\ \varphi_2(i,g) &= \begin{cases} (g, i+1) & \text{if } i < k \\ (g,1) & \text{if } i = k \end{cases} \\ \varphi_3(i,u,g) &= \begin{cases} x_0 & \text{if } i < k \\ \psi_2(u) & \text{if } i = k \end{cases} \\ \varphi_4(i,u,v,g) &= \begin{cases} z & \text{if } i < k \\ u & \text{if } i = k, \end{cases} \end{aligned}$$

where x_0 is any fixed element of X. It follows that the map

$$(i,u,v,w) \mapsto (i, ug_0^{k-i}, vx_0^{i-1}, w),$$

where $i \in [k]$, $u \in G^i$, $v \in X^{k-i}$, $w \in G^k$, defines an injective homomorphism $\mathbf{R} \to \mathbf{R}'$. Moreover, all the automata $\mathbf{R}_i$, $i = 1, 2, 3$ are aperiodic, in fact $\mathbf{R}_2$ is definite and $\mathbf{R}_3$ is an identity-reset automaton. (Alternatively, one may refer to the Krohn-Rhodes theorem by showing that each of the automata $\mathbf{R}_i$ can be embedded in a cascade composition of $\mathbf{U}$ with itself.) □

COROLLARY 6.2 *If $\mathcal{C}$ is a Conway category satisfying the identity $\mathbf{P}_k$, then $\mathcal{C} \models \Gamma(\mathbf{R})$.*

Proof. This is immediate from Lemmas 6.1, 3.5 and 3.6. □

Since G is a subgroup of S, there exists a nonempty set $Q_G \subseteq Q$ which is closed under the functions in G and such that (Q_G, G), equipped with the natural action, is a transformation group having a faithful action. See [11]. Thus, each $g \in G$ defines a permutation $Q_G \to Q_G$, moreover, the unit element of G defines the identity function $Q_G \to Q_G$, and finally, for all $g_1, g_2 \in G$ we have $g_1 = g_2$ if and only if $qg_1 = qg_2$, for all $q \in Q_G$.

Now let $\mathbf{M}$ be the cascade composition

$$\mathbf{M} = \mathbf{R} \times \mathbf{Q}[G, \varphi_1, \varphi_2]$$

determined by the identity function $\varphi_1 : G \longrightarrow G$ and the function $\varphi_2 : R \times G \longrightarrow X$,

$$\varphi_2((i,u,v,w),g) \quad = \quad \begin{cases} x & \text{if } v = xv' \text{ and } x \in X \\ u\psi_1 & \text{if } v = \lambda. \end{cases}$$

(Note that the definition of φ_2 does not depend on g.) Let $\mathbf{M}' = (M', G)$ be the subautomaton of $\mathbf{M}$ determined by those states

$$((i,u,v,w),q) \in R \times Q$$

such that there exists a $q_1 \in Q_G$ with $q_1 v' = q$, where $v' \in X^i$ is the word $\text{first}_i(w\psi)$. (Such a state $q_1 \in Q_G$ is unique, since $v'v = w\psi$ induces a permutation of Q_G.) Below we will denote q_1 by q^{-1}. Note also that $qvu = q^{-1}v'vu = q^{-1}wu \in Q_G$.

LEMMA 6.3 *Suppose that $\mathcal{C}$ is a Conway category satisfying $\mathbf{P}_k$ and the identity $\Gamma(\mathbf{Q})$. Then $\mathcal{C} \models \Gamma(\mathbf{M})$ and $\mathcal{C} \models \Gamma(\mathbf{M}')$.*

Proof. This follows from Corollary 6.2, Lemma 3.6 and Lemma 3.5. □

Let $\mathbf{Q}_G$ denote the transformation group (Q_G, G).

LEMMA 6.4 *The automaton $\mathbf{M}'$ is isomorphic to the direct product $\mathbf{R} \times \mathbf{Q}_G$ of $\mathbf{R}$ and $\mathbf{Q}_G$. An isomorphism $h : \mathbf{M}' \longrightarrow \mathbf{R} \times \mathbf{Q}_G$ is given by the map*

$$((i,u,v,w),q) \quad \mapsto \quad ((i,u,v,w),qvu), \quad \textit{all } ((i,u,v,w),q) \in M'.$$

Proof. We have already noted that $qvu = q^{-1}wu \in Q_G$. Also, if $((i,u,v,w),q_1)$ and $((i,u,v,w),q_2)$ are both in M', then $q_1^{-1} \neq q_2^{-1}$, so that $q_1vu = q_1^{-1}wu \neq q_2^{-1}wu = q_2vu$, since w and u induce permutations $Q_G \longrightarrow Q_G$. This proves that h is injective. To see that h is also surjective, suppose that $((i,u,v,w),q') \in R \times Q_G$. Let q_1 be the state in Q_G with $q_1wu = q'$. This state exists, since w and u induce permutations $Q_G \longrightarrow Q_G$. Then let $q = q_1v'$, where $v'v = w\psi$. We have $((i,u,v,w),q) \in M'$ and $h : ((i,u,v,w),q) \mapsto ((i,u,v,w),q')$. It is straightforward to check that h is a homomorphism. □

COROLLARY 6.5 *Suppose that $\mathcal{C}$ is a Conway category satisfying the kth power identity. If $\mathcal{C} \models \Gamma(\mathbf{Q})$, then $\mathcal{C} \models \Gamma(G)$.*

Proof. By Lemma 6.3, we have $\mathcal{C} \models \Gamma(\mathbf{M}')$. Also, by Corollary 6.2, $\mathcal{C} \models \Gamma(\mathbf{R})$. Thus, by Lemma 3.6 and Lemma 6.4, $\mathcal{C} \models \Gamma(\mathbf{Q}_G)$. Since the action of G on Q_G is faithful, $S(\mathbf{Q}_G)$ is isomorphic to G, and thus the automaton (G,G), equipped with the natural self action is isomorphic to a subautomaton of a direct power of $\mathbf{Q}_G$. It follows that $\mathcal{C} \models \Gamma(G)$. □

We are now ready to complete the proof of Theorem 4.2.

Proof of Theorem 4.2. Suppose that $\mathcal{C}$ is a Conway category satisfying the identities in $\Gamma(\mathcal{Q})$ as well as the nth power identity for some $n > 1$. If $\mathbf{Q} \in \mathcal{Q}$, then by Lemma 5.1, $\mathcal{C} \models \Gamma(\mathbf{Q}^1)$. Also, by Lemma 5.3, $\mathcal{C} \models \mathbf{P}_{n^k}$, for all $k \geq 1$. Since for some k all functions in $S(\mathbf{Q}^1)$ are induced by a word of $\mathbf{Q}^1$

of length n^k, by Corollary 6.5 we have $\mathcal{C} \models \Gamma(G)$ for any subgroup G of $S(\mathbf{Q})$. Thus, by Theorem 3.3, $\mathcal{C} \models \Gamma(S(\mathbf{Q}))$. We conclude that $\mathcal{C}$ satisfies the identity associated with the semigroup of any automaton in $\mathcal{Q}$. From this the result follows by Theorem 3.3. □

Proof of Corollary 4.3. One direction is obvious from Theorem 4.2.

For the other direction suppose that we have $\mathcal{C} \models \Gamma(\mathbf{Q})$ for all Conway categories $\mathcal{C}$ with $\mathcal{C} \models \Gamma(\mathcal{Q})$. Let $\mathcal{G}$ denote the class of simple groups dividing the semigroups of the automata in $\mathcal{Q}$. Then, by Theorem 3.3, $\mathcal{C} \models \Gamma(\mathbf{Q})$ holds for all Conway categories $\mathcal{C}$ with $\mathcal{C} \models \Gamma(\mathcal{G})$. Thus, again by Theorem 3.3, any simple group divisor of $S(\mathbf{Q})$ is in $\mathcal{G}$. □

7 Proof of Theorem 4.5

For each $n \geq 3$, consider the automaton $\mathbf{Q}_n = ([n], X)$ such that $X = \{x, y\}$ with x inducing the transposition (12) and y inducing the cyclic permutation $(12\ldots n)$. From Corollary 4.4 we immediately have

COROLLARY 7.1 *The Conway identities and the equations $\Gamma(\mathbf{Q}_n)$, $n \geq 3$ are complete for iteration theories.*

LEMMA 7.2 *For each $n \geq 3$, and for any Conway category $\mathcal{C}$,*

$$\mathcal{C} \models \mathbf{S}_n \quad \Leftrightarrow \quad \mathcal{C} \models \Gamma(\mathbf{Q}_n).$$

Proof. Let $f : A^2 \times C \to A$ in a Conway category $\mathcal{C}$, and let g denote the morphism on the left hand side of the equation defining $\mathbf{S}_n$. Below we will write π_i^n for $\pi_i^{A^n}$ and $!_k$ for $!_{A^k}$. Morphism Δ_2 is the diagonal $\langle \mathrm{id}_A, \mathrm{id}_A \rangle : A \to A^2$. Note that

$$\begin{aligned} f_{\mathbf{Q}_n} \;=\; & \langle !_1 \times f \cdot (\Delta_2 \times !_{n-3} \times \mathrm{id}_C),\ f \cdot (\mathrm{id}_A \times !_1 \times \mathrm{id}_A \times !_{n-3} \times \mathrm{id}_C), \\ & f \cdot (\langle \pi_3^n, \pi_4^n \rangle \times \mathrm{id}_C),\ \ldots,\ f \cdot (\langle \pi_{n-1}^n, \pi_n^n \rangle \times \mathrm{id}_C),\ f \cdot (\langle \pi_n^n, \pi_1^n \rangle \times \mathrm{id}_C) \rangle. \end{aligned}$$

We will show that

$$\begin{aligned} (f_{\mathbf{Q}_n})^\dagger \;=\; & \langle g,\ f \cdot \langle g, (f^\dagger)^{n-2} \cdot \langle g, \mathrm{id}_C \rangle, \mathrm{id}_C \rangle,\ (f^\dagger)^{n-2} \cdot \langle g, \mathrm{id}_C \rangle,\ \ldots \\ & \ldots,\ f^\dagger \cdot \langle g, \mathrm{id}_C \rangle \rangle. \end{aligned} \tag{6}$$

Indeed, by using Sublemma 5.2, one derives

$$\begin{aligned} (f_{\mathbf{Q}_n})^\dagger \;=\; & \langle !_1 \times f \cdot (\Delta_2 \times !_{n-2} \times \mathrm{id}_C),\ f \cdot (\mathrm{id}_A \times !_1 \times \mathrm{id}_A \times !_{n-3} \times \mathrm{id}_C), \\ & f^\dagger \cdot (\pi_4^n \times \mathrm{id}_C),\ \ldots,\ f^\dagger \cdot (\pi_{n-1}^n \times \mathrm{id}_C),\ f^\dagger \cdot (\pi_1^n \times \mathrm{id}_C) \rangle^\dagger. \end{aligned}$$

Thus, again by the Conway identities,

$$\begin{aligned} (f_{\mathbf{Q}_n})^\dagger \;=\; & \langle !_1 \times f \cdot (\Delta_2 \times !_{n-2} \times \mathrm{id}_C),\ f \cdot (\mathrm{id}_A \times !_1 \times \mathrm{id}_A \times !_{n-3} \times \mathrm{id}_C), \\ & (f^\dagger)^{n-2} \cdot (\pi_1^n \times \mathrm{id}_C),\ \ldots,\ f^\dagger \cdot (\pi_1^n \times \mathrm{id}_C) \rangle^\dagger \\ =\; & \langle g,\ f \cdot \langle g, (f^\dagger)^{n-2} \cdot \langle g, \mathrm{id}_C \rangle, \mathrm{id}_C \rangle,\ (f^\dagger)^{n-2} \cdot \langle g, \mathrm{id}_C \rangle,\ \ldots,\ f^\dagger \cdot \langle g, \mathrm{id}_C \rangle \rangle. \end{aligned}$$

Thus, if $\mathbf{S}_n$ holds in $\mathcal{C}$, then

$$\pi_1^n \cdot (f_{\mathbf{Q}_n})^\dagger = (f \cdot (\Delta_2 \times \mathrm{id}_C))^\dagger = f^{\dagger\dagger}.$$

But then,

$$\begin{aligned} f^\dagger \cdot \langle g, \mathrm{id}_C \rangle &= f^\dagger \cdot \langle f^{\dagger\dagger}, \mathrm{id}_C \rangle \\ &= f^{\dagger\dagger} \end{aligned}$$

and by induction,

$$(f^\dagger)^i \cdot \langle g, \mathrm{id}_C \rangle = f^{\dagger\dagger},$$

for all $i \geq 1$. Thus, also

$$\begin{aligned} f \cdot \langle g, (f^\dagger)^{n-2} \cdot \langle g, \mathrm{id}_C \rangle, \mathrm{id}_C \rangle &= f \cdot \langle f^{\dagger\dagger}, f^{\dagger\dagger}, \mathrm{id}_C \rangle \\ &= f \cdot (\Delta_2 \times \mathrm{id}_C) \cdot \langle (f \cdot (\Delta_2 \times \mathrm{id}_C))^\dagger, \mathrm{id}_C \rangle \\ &= (f \cdot (\Delta_2 \times \mathrm{id}_C))^\dagger \\ &= f^{\dagger\dagger}. \end{aligned}$$

Thus, if $\mathcal{C} \models \mathbf{S}_n$, then, by (6),

$$(f_{\mathbf{Q}_n})^\dagger = \Delta_n \cdot (f \cdot (\Delta_2 \times \mathrm{id}_C))^\dagger = f^{\dagger\dagger},$$

proving $\mathcal{C} \models \Gamma(\mathbf{Q}_n)$. The converse implication is now obvious. □

Proof of Theorem 4.5. By Corollary 7.1, the Conway identities and the equations $\Gamma(\mathbf{Q}_n)$, $n \geq 3$ are complete. But by Lemma 7.2, in Conway categories each identity $\Gamma(\mathbf{Q}_n)$ is equivalent to the equation $\mathbf{S}_n$. □

References

[1] H. Bekič, Definable operations in general algebras, and the theory of automata and flowcharts, *Technical Report*, IBM Laboratory, Vienna, 1969.

[2] L. Bernátsky and Z. Ésik, Semantics of flowchart programs and the free Conway theories. *Theoretical Informatics and Applications*, 32(1998), 35–78.

[3] S.L. Bloom and Z. Ésik, *Iteration Theories: The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1993.

[4] S.L. Bloom and Z. Ésik, Fixed point operations on ccc's. Part 1. *Theoretical Computer Science*, 155(1996), 1–38.

[5] S.L. Bloom and Z. Ésik, The equational logic of fixed points. *Theoretical Computer Science*, 179(1997), 1–60.

[6] J.C. Conway, *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.

[7] S. Eilenberg, *Automata, Languages, and Machines*, vol. B. Academic Press, 1976.

[8] Z. Ésik, Identities in iterative and rational algebraic theories. *Computational Linguistics and Computer Languages*, 14(1980), 183-207.

[9] Z. Ésik, Group axioms for iteration. *Information and Computation.* To appear.

[10] Z. Ésik, The power of the group axioms for iteration. *International J. Algebra and Computation.* To appear.

[11] F. Gécseg, *Products of Automata*, Springer, 1986.

[12] G. Lallement, *Semigroups and Combinatorial Applications.* Wiley-Interscience, 1979.

Acta Cybernetica **14** (1999) 83-98.

On One-Pass Term Rewriting

Zoltán Fülöp * Eija Jurvanen † Magnus Steinby ‡
Sándor Vágvölgyi§

Dedicated to Professor Ferenc Gécseg on the occasion of his 60th birthday.

Abstract

Two restricted ways to apply a term rewriting system (TRS) to a tree are considered. When the *one-pass root-started* strategy is followed, rewriting starts from the root and continues stepwise towards the leaves without ever rewriting a part of the current tree produced in a previous rewrite step. *One-pass leaf-started rewriting* is defined similarly, but rewriting begins from the leaves. In the *sentential form inclusion problem* one asks whether all trees which can be obtained with a given TRS from the trees of some regular tree language T belong to another given regular tree language U, and in the *normal form inclusion problem* the same question is asked about the normal forms of T. We show that for a left-linear TRS these problems can be decided for both of our one-pass strategies. In all four cases the decision algorithm involves the construction of a suitable tree recognizer.

1 Introduction

In general, reducing a term with a term rewriting system (TRS) is a highly nondeterministic process in which many choices have to made, and usually no bound for the lengths of the possible reduction sequences can be given in advance. In this paper we consider two very restrictive strategies of term rewriting, *one-pass root-started rewriting* and *one-pass leaf-started rewriting*. When the former strategy is followed, rewriting starts at the root of the given term t and proceeds continuously towards the leaves without ever rewriting any part of the current term which has

*József Attila University, Department of Computer Science, H-6701 Szeged, P. O. Box 652, Hungary, Email: fulop@inf. u-szeged.hu

†Turku Centre for Computer Science, DataCity, Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland, Email: jurvanen@utu.fi

‡Turku Centre for Computer Science, and Department of Mathematics, University of Turku, FIN-20014 Turku, Finland, Email: steinby@utu.fi

§József Attila University, Department of Applied Informatics, H-6701 Szeged, P. O. Box 652, Hungary, Email: vagvolgy@inf.u-szeged.hu

been produced in a previous rewrite step. When no more rewriting is possible, a *one-pass root-started normal form* of the original term t has been reached. Of course, such a normal form is not necessarily irreducible in the usual sense since a rewrite rule may apply either in the part already rewritten or to a subtree rooted at a position strictly below the nodes affected by the last rewriting steps. The leaf-started version is similar, but the rewriting is initiated at the leaves and proceeds towards the root. The requirement that rewriting should always concern positions immediately adjacent to parts of the term rewritten in previous steps distinguishes our rewriting strategies from the IO and OI rewriting schemes considered in [ES77, ES78] or [DDC87]. It also implies that the top-down and bottom-up cases are different even for a linear TRS.

Both of the one-pass modes of rewriting are defined formally by associating with any given TRS an auxiliary TRS in which a new separator mark restricts rewriting in the intended way.

Let us now describe the problems concerning one-pass rewriting considered in this paper. Since the problems involve regular tree languages, we find it convenient to use the terminology of the theory of tree languages. Let $\mathcal{R} = (\Sigma, R)$ be a TRS over a ranked alphabet Σ. For any Σ-tree language T ($\subseteq T_\Sigma$), we denote the sets of one-pass root-started sentential forms, one-pass root-started normal forms, one-pass leaf-started sentential forms and one-pass leaf-started normal forms of trees in T by $1r\mathrm{S}_\mathcal{R}(T)$, $1r\mathrm{N}_\mathcal{R}(T)$, $1\ell\,\mathrm{S}_\mathcal{R}(T)$ and $1\ell\,\mathrm{N}_\mathcal{R}(T)$, respectively. We show that all of the following inclusion problems, in which the input consists of a left-linear TRS $\mathcal{R} = (\Sigma, R)$ and two regular Σ-tree languages T_1 and T_2 (effectively given by tree recognizers, for example), are decidable.

The one-pass root-started sentential form inclusion problem: $1r\mathrm{S}_\mathcal{R}(T_1) \subseteq T_2$?

The one-pass root-started normal form inclusion problem: $1r\mathrm{N}_\mathcal{R}(T_1) \subseteq T_2$?

The one-pass leaf-started sentential form inclusion problem: $1\ell\,\mathrm{S}_\mathcal{R}(T_1) \subseteq T_2$?

The one-pass leaf-started normal form inclusion problem: $1\ell\,\mathrm{N}_\mathcal{R}(T_1) \subseteq T_2$?

In [GT95] the sentential form inclusion problem for ordinary sentential forms is called the second-order reachability problem and the problem is shown to be decidable for a TRS $\mathcal{R}$ which preserves recognizability, i.e. if the set $\mathrm{S}_\mathcal{R}(T)$ of sentential forms of the trees of any recognizable tree language T is also recognizable.

Many questions concerning term rewriting systems have been studied, and solved, using tree automata and tree languages; cf. [DDC87, DG89, Gil91, GT95, GV98, HH94, KT95, VG92], for example. Also here tree automata are used: in all four cases the decidability of the problem is proved by showing how one may construct from $\mathcal{R}$ and the given tree recognizers of T_1 and T_2 a new tree recognizer $\mathcal{C}$ such that the question can be answered by checking whether the tree language $T(\mathcal{C})$ recognized by $\mathcal{C}$ is empty. To simplify these constructions we introduce generalized top-down and generalized bottom-up tree recognizers. It is easy to see that both of these new types of recognizers recognize exactly the regular tree languages and that their emptiness problems are decidable.

The paper is essentially self-contained since all special concepts used, as well as many general notions, are completely defined. However, for further information about term rewriting systems and tree automata, we refer the reader to [Ave95], [DJ90], [Hue80], [GS84] and [GS97].

A conference version of this paper has appeared as [FJSV98]. This research was supported by the exchange program of the University of Turku and the József Attila University, and by the grants MKM 665/96 and FKFP 0095/97.

2 Preliminaries

Throughout this paper Σ is a *ranked alphabet*, i.e. a finite set of operation symbols. For each $m \geq 0$, the set of m-ary symbols in Σ is denoted by Σ_m. We say that Σ is *unary* if $\Sigma = \Sigma_1$, i.e., if every symbol $f \in \Sigma$ has rank 1. If Y is an alphabet disjoint with Σ, then the set $T_\Sigma(Y)$ of *Σ-terms with variables* in Y is the smallest set U of strings such that

(1) $Y \cup \Sigma_0 \subseteq U$ and

(2) $f(t_1, \ldots, t_m) \in U$ whenever $m \geq 1$, $f \in \Sigma_m$ and $t_1, \ldots, t_m \in U$.

Sometimes we consider terms $f(t_1, \ldots, t_m)$ with $m \geq 0$. For $m = 0$, this is interpreted as f. The set $T_\Sigma(\emptyset)$ of *ground Σ-terms* is denoted by T_Σ. Terms are viewed in the usual way as formal representations of trees, and we also call them trees. In particular, ground Σ-terms and subsets of T_Σ are called *Σ-trees* and *Σ-tree languages*, respectively.

The *height* $\mathrm{hg}(t)$ of a tree $t \in T_\Sigma(Y)$ is defined so that $\mathrm{hg}(t) = 0$ for $t \in Y \cup \Sigma_0$, and $\mathrm{hg}(t) = \max\{\mathrm{hg}(t_1), \ldots, \mathrm{hg}(t_m)\} + 1$ for $t = f(t_1, \ldots, t_m)$. The set $\mathrm{var}(t)$ $(\subseteq Y)$ of variables appearing in t is also defined as usual (cf. [GS97], for example).

Let $X = \{x_1, x_2, \ldots\}$ be a countably infinite set of variables. For every $n \geq 0$, we put $X_n = \{x_1, \ldots, x_n\}$ and abbreviate $T_\Sigma(X_n)$ to $T_{\Sigma,n}$. A tree $t \in T_{\Sigma,n}$ is called *linear* if each x_i, $1 \leq i \leq n$, appears at most once in t.

We introduce a subset $\widetilde{T}_{\Sigma,n}$ of $T_{\Sigma,n}$ as follows. A tree $t \in T_{\Sigma,n}$ belongs to $\widetilde{T}_{\Sigma,n}$ if and only if each of $x_1, \ldots, x_n$ occurs in t exactly once and their left-to-right order is $x_1, \ldots, x_n$. Hence the elements of $\widetilde{T}_{\Sigma,n}$ are special linear trees. In addition, let $\widetilde{T}_{\Sigma,X} = \bigcup_{n=0}^{\infty} \widetilde{T}_{\Sigma,n}$. If $f \in \Sigma_m$, $m \geq 1$ and $t_1, \ldots, t_m \in \widetilde{T}_{\Sigma,X}$, then $\|f(t_1, \ldots, t_m)\|$ is the unique tree in $\widetilde{T}_{\Sigma,X}$ obtained from $f(t_1, \ldots, t_m)$ by renaming the variables.

A *substitution* $\sigma : X \to T_\Sigma(X)$ is extended to a mapping $\sigma : T_\Sigma(X) \to T_\Sigma(X)$ so that $\sigma(t) = f(\sigma(t_1), \ldots, \sigma(t_m))$ for any $t = f(t_1, \ldots, t_m)$. Hence, for any tree $t \in T_\Sigma(X)$, $\sigma(t)$ is the tree obtained from t when every occurrence of each variable $x \in \mathrm{var}(t)$ is replaced by the corresponding tree $\sigma(x)$. If, in particular, $t \in T_{\Sigma,n}$ and $\sigma(x_i) = t_i$ $(i = 1, 2, \ldots, n)$, we write $\sigma(t) = t[t_1, \ldots, t_n]$.

Let Σ be a ranked alphabet. A *term rewriting system* (TRS, for short) over Σ is a system $\mathcal{R} = (\Sigma, R)$, where R is a finite subset of $T_\Sigma(X) \times T_\Sigma(X)$ such that $\mathrm{var}(r) \subseteq \mathrm{var}(p)$ and $p \notin X$ for each $(p, r) \in R$. Note that since R is finite, $R \subseteq T_\Sigma(X_n) \times T_\Sigma(X_n)$ for some $n \geq 0$. The elements of R are called (*rewrite*)

rules and written in the form $p(x_1, \ldots, x_n) \to r(x_1, \ldots, x_n)$ or just as $p \to r$. A rule $p \to r$ of a TRS $\mathcal{R}$ is called *ground* if both p and r are ground trees.

A TRS $\mathcal{R}$ induces a binary relation $\Rightarrow_\mathcal{R}$ in T_Σ defined as follows: for any t, $u \in T_\Sigma$, $t \Rightarrow_\mathcal{R} u$ if u is obtained from t by replacing an occurrence of a subtree of t of the form $p[t_1, \ldots, t_n]$ by $r[t_1, \ldots, t_n]$, where $p \to r \in R$ and $t_1, \ldots, t_n \in T_\Sigma$. The reflexive, transitive closure of $\Rightarrow_\mathcal{R}$ is denoted by $\Rightarrow_\mathcal{R}^*$. Hence $s \Rightarrow_\mathcal{R}^* t$ if and only if there exists a *reduction sequence*

$$t_0 \Rightarrow_\mathcal{R} t_1 \Rightarrow_\mathcal{R} \cdots \Rightarrow_\mathcal{R} t_n$$

in $\mathcal{R}$ such that $n \geq 0$, $t_0 = s$ and $t_n = t$.

A TRS $\mathcal{R}$ is *left-linear* if, for every rule $p \to r$ in R, p is a linear tree. A TRS is *in standard form* if for every rule $p \to r$ in it, $p \in \widetilde{T}_{\Sigma,n}$ for some $n \geq 0$. Obviously one can construct for every left-linear TRS $\mathcal{R}$ a standard form TRS $\mathcal{R}'$ such that $\Rightarrow_\mathcal{R} = \Rightarrow_{\mathcal{R}'}$.

We define the set of *left-hand sides* of a TRS $\mathcal{R} = (\Sigma, R)$ as $\mathrm{lhs}(\mathcal{R}) = \{\, p \mid (\exists r)\, p \to r \in R \,\}$.

An element $s \in T_\Sigma$ is *irreducible* with respect to $\mathcal{R}$ if there exists no u such that $s \Rightarrow_\mathcal{R} u$. A Σ-tree s is a *normal form* of a Σ-tree t if $t \Rightarrow_\mathcal{R}^* s$ and s is irreducible. The set of all normal forms of a Σ-tree t is denoted by $\mathrm{N}_\mathcal{R}(t)$. The set of *sentential forms* of t is defined by

$$\mathrm{S}_\mathcal{R}(t) = \{\, s \mid t \Rightarrow_\mathcal{R}^* s \,\}.$$

Moreover, for a tree language $T \subseteq T_\Sigma$, we put

$$\mathrm{S}_\mathcal{R}(T) = \bigcup_{t \in T} \mathrm{S}_\mathcal{R}(t),$$
$$\mathrm{N}_\mathcal{R}(T) = \bigcup_{t \in T} \mathrm{N}_\mathcal{R}(t).$$

A TRS $\mathcal{R}$ over Σ is *terminating* if there are no infinite reduction sequences $t_1 \Rightarrow_\mathcal{R} t_2 \Rightarrow_\mathcal{R} \cdots$, cf. [Hue80], [DJ90] and [Ave95].

Let us recall the two basic types of tree recognizers which both define the same family of recognizable tree languages. To facilitate our proofs, we also introduce certain generalized versions of both types. All of these recognizers can be defined conveniently as special term rewriting systems.

In a *top-down* Σ*-recognizer* $\mathcal{A} = (A, \Sigma, P, a_0)$

(1) A is a (finite) unary ranked alphabet of *states* such that $A \cap \Sigma = \emptyset$,

(2) P is a finite set of rewrite rules, the *transition rules*, each of the form

   (a) $a(f(x_1, \ldots, x_m)) \to f(a_1(x_1), \ldots, a_m(x_m))$, also written simply $a(f) \to f(a_1, \ldots, a_m)$, where $m > 0$, $f \in \Sigma_m$ and $a, a_1, \ldots, a_m \in A$, or of the form

   (b) $a(c) \to c$, where $c \in \Sigma_0$ and $a \in A$, and

(3) $a_0 \in A$ is the *initial state.*

We treat $\mathcal{A}$ as the TRS $(\Sigma \cup A, P)$ and the rewrite relation $\Rightarrow_{\mathcal{A}} \subseteq T_{\Sigma \cup A} \times T_{\Sigma \cup A}$ is defined accordingly. For each $a \in A$, let $T(\mathcal{A}, a) = \{ t \in T_\Sigma \mid a(t) \Rightarrow_{\mathcal{A}}^* t \}$. In particular, the tree language *recognized* by $\mathcal{A}$ is the set $T(\mathcal{A}) = T(\mathcal{A}, a_0)$.

A tree language $T \subseteq T_\Sigma$ is *recognizable* or *regular* if there exists a top-down Σ-recognizer $\mathcal{A}$ such that $T(\mathcal{A}) = T$. The class of all recognizable Σ-tree languages is denoted by REC_Σ.

In a *generalized top-down* Σ*-recognizer* $\mathcal{A} = (A, \Sigma, P, a_0)$

(1) A is a (finite) unary ranked alphabet of *states* such that $A \cap \Sigma = \emptyset$,

(2) P is a finite set of rewrite rules, the *transition rules* of $\mathcal{A}$, of the form

$$a(t(x_1, \ldots, x_n)) \to t[a_1(x_1), \ldots, a_n(x_n)],$$

where $n \geq 0$, $a, a_1, \ldots, a_n \in A$, and $t \in \widetilde{T}_{\Sigma,n}$, and

(3) $a_0 \in A$ is the *initial state.*

The rewrite relation $\Rightarrow_{\mathcal{A}} \subseteq T_{\Sigma \cup A} \times T_{\Sigma \cup A}$ and its reflexive, transitive closure $\Rightarrow_{\mathcal{A}}^*$ are defined in the natural way. The tree language *recognized* by $\mathcal{A}$ is the set

$$T(\mathcal{A}) = \{ t \in T_\Sigma \mid a_0(t) \Rightarrow_{\mathcal{A}}^* t \}.$$

It is clear that also the generalized top-down Σ-recognizers recognize exactly the regular Σ-tree languages.

Next we define tree recognizers which read their inputs from the leaves to the root.

A *bottom-up* Σ*-recognizer* is a quadruple $\mathcal{A} = (A, \Sigma, P, A_f)$, where

(1) A is a finite set of *states* of rank 0, $\Sigma \cap A = \emptyset$,

(2) P is a finite set of rewrite rules of the form

$$f(a_1, \ldots, a_m) \to a$$

with $m \geq 0$, $f \in \Sigma_m$, $a_1, \ldots, a_m, a \in A$, and

(3) A_f $(\subseteq A)$ is the set of *final states.*

We say that $\mathcal{A}$ is *total deterministic* if for all $f \in \Sigma_m$, $m \geq 0$, $a_1, \ldots, a_m \in A$, there is exactly one rule of the form $f(a_1, \ldots, a_m) \to a$.

When we treat $\mathcal{A}$ as the rewriting system $(\Sigma \cup A, P)$, the tree language recognized by it can be defined as the set

$$T(\mathcal{A}) = \{ t \in T_\Sigma \mid (\exists\, a \in A_f)\ t \Rightarrow_{\mathcal{A}}^* a \}.$$

For any bottom-up Σ-recognizer $\mathcal{A}$, one can effectively construct a total deterministic bottom-up Σ-recognizer $\mathcal{B}$ such that $T(\mathcal{A}) = T(\mathcal{B})$. If $\mathcal{A}$ is total deterministic, there is for each Σ-tree t exactly one state $a \in A$ such that $t \Rightarrow_{\mathcal{A}}^* a$.

A *generalized bottom-up* Σ-*recognizer* is a system $\mathcal{A} = (A, \Sigma, P, A_f)$, where A, Σ and A_f are as in the case of a bottom-up Σ-recognizer, but P is now a finite set of rewrite rules of the form

$$t[a_1, \ldots, a_n] \to a,$$

where $n \geq 0$, $t \in \widetilde{T}_{\Sigma,n}$ and $a_1, \ldots, a_n$, $a \in A$. Hence there may be rules of the form $a \to b$ in P, where a, $b \in A$. The tree language recognized by $\mathcal{A}$ is $T(\mathcal{A}) = \{\, t \in T_\Sigma \mid (\exists\, a \in A_f)\ t \Rightarrow_{\mathcal{A}}^* a \,\}$.

It is not hard to see that the class of Σ-tree languages recognized by generalized bottom-up Σ-recognizers is REC_Σ. Moreover, the emptiness problem "$T(\mathcal{A}) = \emptyset$?" is obviously decidable for both generalized top-down and generalized bottom-up recognizers.

3 One-pass Term Rewriting

The first of our two modes of one-pass rewriting may be described as follows.

Let $\mathcal{R} = (\Sigma, R)$ be a TRS and t the Σ-tree to be rewritten. The rewriting starts at the root of t and the portion first rewritten should include the root. Rewriting then proceeds as far as possible towards the leaves so that each rewrite step applies to a root segment of some maximal unprocessed subtree but never involves any part of the tree produced by a previous rewrite step. For the formal definition we associate with $\mathcal{R}$ a TRS in which a new special symbol forces this mode of rewriting.

Definition 3.1. The *one-pass root-started TRS* associated with a given TRS $\mathcal{R} = (\Sigma, R)$ is the TRS $\mathcal{R}_\# = (\Sigma \cup \{\#\}, R_\#)$, where $\#$ is a new unary symbol, the *separator mark*, and $R_\#$ is the set of all rewrite rules

$$\#(p(x_1, \ldots, x_n)) \to r[\#(x_1), \ldots, \#(x_n)]$$

obtained from a rule $p(x_1, \ldots, x_n) \to r(x_1, \ldots, x_n)$ in R by adding $\#$ to the root of the left-hand side and above the variables in the right-hand side.

Example 3.2. Suppose that Σ consists of the binary symbol f, the unary symbol g and the nullary symbol c. If $\mathcal{R} = (\Sigma, R)$ is the TRS, where

$$R = \{\, f(g(x_1), x_2) \to f(x_1, g(x_2)),\ g(x_1) \to f(c, x_1),\ g(x_1) \to g(c) \,\},$$

then the rule set of the associated one-pass root-started TRS $\mathcal{R}_\#$ is

$$\begin{aligned} R_\# = \{\, & \#(f(g(x_1), x_2)) \to f(\#(x_1), g(\#(x_2))), \\ & \#(g(x_1)) \to f(c, \#(x_1)),\ \#(g(x_1)) \to g(c) \,\}. \end{aligned}$$

For any TRS $\mathcal{R}$, the associated one-pass root-started TRS $\mathcal{R}_\#$ is terminating. For recovering the one-pass root-started reduction sequences of $\mathcal{R}$ from the reduction sequences of $\mathcal{R}_\#$, we introduce the tree homomorphism $\delta : T_{\Sigma\cup\{\#\}} \to T_\Sigma$ which erases the separator marks:

(1) $\delta(c) = c$ for any $c \in \Sigma_0$;

(2) $\delta(\#(t)) = \delta(t)$ for any $t \in T_{\Sigma\cup\{\#\}}$;

(3) $\delta(t) = f(\delta(t_1), \ldots, \delta(t_m))$ for $t = f(t_1, \ldots, t_m)$, where $m > 0$, $f \in \Sigma_m$ and $t_i \in T_{\Sigma\cup\{\#\}}$.

If $t \in T_\Sigma$ and

$$\#(t) \Rightarrow_{\mathcal{R}_\#} t_1 \Rightarrow_{\mathcal{R}_\#} t_2 \Rightarrow_{\mathcal{R}_\#} \ldots \Rightarrow_{\mathcal{R}_\#} t_k$$

is a reduction sequence with $\mathcal{R}_\#$, then

$$t \Rightarrow_{\mathcal{R}} \delta(t_1) \Rightarrow_{\mathcal{R}} \delta(t_2) \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} \delta(t_k)$$

is a *one-pass root-started reduction sequence* with $\mathcal{R}$. The terms t, $\delta(t_1)$, ... , $\delta(t_k)$ are called *one-pass root-started sentential forms* of t in $\mathcal{R}$. If t_k is irreducible in $\mathcal{R}_\#$, then $\delta(t_k)$ is a *one-pass root-started normal form* of t in $\mathcal{R}$. The sets of all one-pass root-started sentential forms and normal forms of a Σ-tree t are denoted by $1r\mathrm{S}_{\mathcal{R}}(t)$ and $1r\mathrm{N}_{\mathcal{R}}(t)$, respectively. This notation is extended to sets of Σ-trees in the natural way.

Note that for any TRS $\mathcal{R} = (\Sigma, R)$ and any $t \in T_\Sigma$, the sets $1r\mathrm{S}_{\mathcal{R}}(t)$ and $1r\mathrm{N}_{\mathcal{R}}(t)$ are finite and effectively computable but that $1r\mathrm{S}_{\mathcal{R}}(T)$ and $1r\mathrm{N}_{\mathcal{R}}(T)$ are not necessarily regular even for a regular Σ-tree language T.

The mode of one-pass rewriting which starts from the leaves is also formalized by associating with the given TRS a special one-pass TRS. This TRS is constructed in two stages. First we add to the original TRS all rules obtained by instantiating any variables in the original rules as constants. In the second stage the extended TRS is turned into a one-pass TRS by introducing a separator mark and by labelling the symbols of the right-hand sides so that the rewriting cannot be restarted from leaves which have already been processed.

Definition 3.3. Let $\mathcal{R} = (\Sigma, R)$ be a TRS. First we extend R to the set R_e of all rules

$$p[y_1, \ldots, y_n] \to q[y_1, \ldots, y_n]$$

such that $p(x_1, \ldots, x_n) \to q(x_1, \ldots, x_n) \in R$, with p, $q \in T_{\Sigma,n}$, and for each i, $1 \leq i \leq n$, either $y_i \in X$ or $y_i \in \Sigma_0$, and $p[y_1, \ldots, y_n] \in \widetilde{T}_{\Sigma,X}$. Now let $\Sigma' = \{ f' \mid f \in \Sigma \}$ be a disjoint copy of Σ such that for any $f \in \Sigma$, f and f' have the same rank. The *one-pass leaf-started TRS* associated with $\mathcal{R}$ is the TRS $\mathcal{R}^\# = (\Sigma \cup \Sigma' \cup \{\#\}, R^\#)$, where $\#$ is a new unary symbol, the *separator mark*, and $R^\#$ consists of all rules

$$p[\#(x_1), \ldots, \#(x_n)] \to \#(r'(x_1, \ldots, x_n)),$$

where $p \to r \in R_e$, with p, $r \in T_{\Sigma,n}$, and r' is obtained from r by replacing every symbol $f \in \Sigma$ by the corresponding symbol f' in Σ'.

For every left-linear TRS $\mathcal{R}$, the one-pass TRS $\mathcal{R}^\#$ is in standard form.

Example 3.4. Let $\Sigma = \{f, g, c\}$, where f is binary, g unary and c nullary, and let $\mathcal{R} = (\Sigma, R)$ be the TRS with

$$R = \{\, f(g(x_1), x_2) \to f(x_1, g(x_2)),\ g(c) \to f(c, c) \,\}.$$

Then $\Sigma' = \{f', g', c'\}$ and the one-pass leaf-started TRS associated with R is the TRS $\mathcal{R}^{\#} = (\Sigma \cup \Sigma' \cup \{\#\}, R^{\#})$ where $R^{\#}$ is

$$\begin{aligned} R^{\#} = \{\, & f(g(\#(x_1)), \#(x_2)) \to \#(f'(x_1, g'(x_2))), \\ & f(g(c), \#(x_1)) \to \#(f'(c', g'(x_1))), \\ & f(g(\#(x_1)), c) \to \#(f'(x_1, g'(c'))), \\ & f(g(c), c) \to \#(f'(c', g'(c'))),\ g(c) \to \#(f'(c', c')) \,\}. \end{aligned}$$

It is clear that the TRS $\mathcal{R}_e = (\Sigma, R_e)$ is always equivalent to the original TRS $\mathcal{R}$ in the sense that $\Rightarrow_{\mathcal{R}_e} = \Rightarrow_{\mathcal{R}}$. The connection between $\mathcal{R}$ and $\mathcal{R}^{\#}$ is the following. The reduction sequences of $\mathcal{R}^{\#}$ represent such reduction sequences of $\mathcal{R}$ which start at the leaves of a term and proceed towards the root of it in such a way that symbols introduced by a previous rewrite step never form a part of the left-hand side of the rule applied next. At the same time the rewrite places are joined together by the separator mark. Moreover, $\mathcal{R}^{\#}$ can make only one pass over the term because the left-hand sides and the right-hand sides of its rules are over disjoint ranked alphabets. The one-pass reduction sequence of $\mathcal{R}$ represented by a reduction sequence of $\mathcal{R}^{\#}$ is recovered by applying a tree homomorphism which erases the #-marks and the primes from the symbols $f' \in \Sigma'$. Formally we define $\delta\colon T_{\Sigma \cup \Sigma' \cup \{\#\}} \to T_{\Sigma}$ so that

(1) $\delta(c') = \delta(c) = c$ for every $c \in \Sigma_0$;

(2) $\delta(\#(t)) = \delta(t)$ for every $t \in T_{\Sigma \cup \Sigma' \cup \{\#\}}$;

(3) $\delta(f(t_1, \ldots, t_m)) = \delta(f'(t_1, \ldots, t_m)) = f(\delta(t_1), \ldots, \delta(t_m))$ for any $f \in \Sigma_m$, $m \geq 1$, and $t_1, \ldots, t_m \in T_{\Sigma \cup \Sigma' \cup \{\#\}}$.

Then each reduction sequence

$$t \Rightarrow_{\mathcal{R}^{\#}} t_1 \Rightarrow_{\mathcal{R}^{\#}} t_2 \Rightarrow_{\mathcal{R}^{\#}} \ldots \Rightarrow_{\mathcal{R}^{\#}} t_k$$

with $\mathcal{R}^{\#}$ starting from a Σ-tree t yields the *one-pass leaf-started reduction sequence*

$$t \Rightarrow_{\mathcal{R}} \delta(t_1) \Rightarrow_{\mathcal{R}} \delta(t_2) \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} \delta(t_k)$$

with $\mathcal{R}$. Moreover, the *set of one-pass leaf-started sentential forms* and the *set of one-pass leaf-started normal forms* of t with respect to $\mathcal{R}$ are defined by

$$1\ell \mathrm{S}_{\mathcal{R}}(t) = \{\, \delta(s) \mid s \in \mathrm{S}_{\mathcal{R}^{\#}}(t) \,\},$$

and

$$1\ell \mathrm{N}_{\mathcal{R}}(t) = \{\, \delta(s) \mid s \in \mathrm{N}_{\mathcal{R}^{\#}}(t) \,\},$$

respectively. Finally, for a tree language $T \subseteq T_\Sigma$, we put

$$1\ell S_\mathcal{R}(T) = \bigcup_{t \in T} 1\ell S_\mathcal{R}(t),$$

and

$$1\ell N_\mathcal{R}(T) = \bigcup_{t \in T} 1\ell N_\mathcal{R}(t).$$

The extended TRS $\mathcal{R}_e$ may seem redundant, but without the new rules in $R_e \setminus R$ many natural one-pass leaf-started rewriting sequences would be missed. In particular, if $\mathcal{R}$ contains no ground rules, such as the rule $g(c) \to f(c,c)$, no non-trivial one-pass leaf-started rewriting sequence could be initiated since the left-hand sides of all rules of $\mathcal{R}^\#$ would contain the symbol #. Hence we would have $1\ell S_\mathcal{R}(t) = 1\ell N_\mathcal{R}(t) = \{t\}$ for every $t \in T_\Sigma$.

4 The one-pass root-started inclusion problems

First we define the one-pass root-started normal form inclusion problem for a TRS $\mathcal{R} = (\Sigma, R)$. It is assumed that the recognizable tree languages are given in the form of tree recognizers.

Theorem 4.1. For any left-linear TRS $\mathcal{R} = (\Sigma, R)$, the following *one-pass root-started normal form inclusion problem* is decidable.
Instance: Recognizable Σ-tree languages T_1 and T_2.
Question: $1rN_\mathcal{R}(T_1) \subseteq T_2$?

For proving Theorem 4.1, we need the following auxiliary notations. For a set A of unary symbols such that $A \cap \Sigma = \emptyset$ and any alphabet Y, let $T_\Sigma(A(Y))$ be the least subset T of $T_{\Sigma \cup A}(Y)$ for which

(1) $\Sigma_0 \subseteq T$,

(2) $a(y) \in T$ for all $a \in A$, $y \in Y$, and

(3) $m \geq 1$, $f \in \Sigma_m$, $t_1, \ldots, t_m \in T$ implies $f(t_1, \ldots, t_m) \in T$.

Let $\mathcal{A} = (A, \Sigma, P, a_0)$ be a top-down Σ-recognizer. For any $a \in A$, $n \geq 0$ and any $t \in T_{\Sigma,n}$, the set $\mathcal{A}(a,t)$ $(\subseteq T_\Sigma(A(X_n)))$ is defined so that

(1) $\mathcal{A}(a, x_i) = \{a(x_i)\}$ for all $x_i \in X_n$,

(2) for $c \in \Sigma_0$, $\mathcal{A}(a,c) = \{c\}$ if $a(c) \to c \in P$, and $\mathcal{A}(a,c) = \emptyset$ otherwise, and

(3) for $m \geq 1$, $t = f(t_1, \ldots, t_m)$,

$$\mathcal{A}(a,t) = \{\, f(s_1, \ldots, s_m) \mid s_1 \in \mathcal{A}(a_1, t_1), \ldots, s_m \in \mathcal{A}(a_m, t_m), \\ a(f) \to f(a_1, \ldots, a_m) \in P \,\}.$$

For any $s \in T_\Sigma(A(X))$ and any variable $x_i \in X$, we denote by $\mathrm{st}(s, x_i)$ the set of states $b \in A$ such that $b(x_i)$ appears as a subterm in s.

Clearly, $\mathcal{A}(a,t) \neq \emptyset$ if and only if there is a computation of $\mathcal{A}$ which starts at the root of t and continues successfully along all paths to the leaves of t, and moreover, if $\mathcal{A}$ reaches in a state b ($\in A$) a leaf labelled by a nullary symbol c, then the rule $b(c) \to c$ is in P. Each term s in $\mathcal{A}(a,t)$ represents the situation when such a successful computation has been completed so that all leaves labelled with a nullary symbol have also been processed. If $t \in \widetilde{T}_{\Sigma,n}$, then every $s \in \mathcal{A}(a_0,t)$ is of the form $s = t[a_1(x_1), \ldots, a_n(x_n)]$ and for any $t_1, \ldots, t_n \in T_\Sigma$, the tree s appears in a computation of $\mathcal{A}$ on $t[t_1, \ldots, t_n]$ of the form

$$a_0(t[t_1, \ldots, t_n]) \Rightarrow_{\mathcal{A}}^* t[a_1(t_1), \ldots, a_n(t_n)] = s[t_1, \ldots, t_n] \Rightarrow_{\mathcal{A}}^* \cdots$$

in which each subterm t_i is processed starting in the corresponding state a_i. However, if t is not linear, then a variable x_i may appear in a term $s \in \mathcal{A}(a_0,t)$ together with more than one state symbol, and then the corresponding subterm t_i should be accepted by a computation starting with each $a \in \mathrm{st}(s, x_i)$.

Proof of Theorem 4.1. Consider a left-linear TRS $\mathcal{R} = (\Sigma, R)$ and any recognizable Σ-tree languages T_1 and T_2. Let $\mathcal{A} = (A, \Sigma, P_1, a_0)$ and $\mathcal{B} = (B, \Sigma, P_2, b_0)$ be top-down Σ-recognizers for which $T(\mathcal{A}) = T_1$ and $T(\mathcal{B}) = T_2^c$ $(= T_\Sigma \setminus T_2)$. We construct a generalized top-down Σ-recognizer $\mathcal{C}$ such that for any Σ-term t,

$$t \in T(\mathcal{C}) \quad \text{iff} \quad t \in T(\mathcal{A}) \text{ and } s \in T(\mathcal{B}) \text{ for some } s \in {}_1r\mathrm{N}_{\mathcal{R}}(t). \qquad (*)$$

Then ${}_1r\mathrm{N}_{\mathcal{R}}(T_1) \subseteq T_2$ if and only if $T(\mathcal{C}) = \emptyset$, and the latter condition is decidable.

Let $\mathcal{C} = (C, \Sigma, P, (a_0, \{b_0\}))$ be the generalized top-down Σ-recognizer with the state set

$$C = (A \times \wp(B)) \cup (\bar{A} \times \wp(B)),$$

where $\wp(B)$ is the power set of B and $\bar{A} = \{\, \bar{a} \mid a \in A \,\}$ is a disjoint copy of A, and the set P of transition rules is defined as follows. The rules are of three different types.

Type 1. If $p(x_1, \ldots, x_n) \to r(x_1, \ldots, x_n)$ is a rule in R and $(a, H) \in A \times \wp(B)$, where $H = \{b_1, \ldots, b_k\}$, we include in P any rule

$$(a,H)(p(x_1, \ldots, x_n)) \to p[(a_1, H_1)(x_1), \ldots, (a_n, H_n)(x_n)]$$

where

(a) $p[a_1(x_1), \ldots, a_n(x_n)] \in \mathcal{A}(a,p)$ and

(b) there are terms $s_1 \in \mathcal{B}(b_1, r)$, $\ldots$, $s_k \in \mathcal{B}(b_k, r)$ such that, for all $i = 1, \ldots, n$,

$$H_i = \mathrm{st}(s_1, x_i) \cup \cdots \cup \mathrm{st}(s_k, x_i).$$

For $H = \emptyset$ ($k = 0$), this is interpreted to mean that $H_1 = \cdots = H_n = \emptyset$ should hold, and if $p \to r$ is a ground rule ($n = 0$), we include $(a, H)(p) \to p$ in P iff $a(p) \Rightarrow_{\mathcal{A}}^* p$ and $b_i(r) \Rightarrow_{\mathcal{B}}^* r$ for all $i = 1, \ldots, k$.

Type 2. Let NI be the set of all terms $q \in \widetilde{T}_{\Sigma,X}$ such that

(1) $\mathrm{hg}(q) \leq \max\{\, \mathrm{hg}(p) \mid p \in \mathrm{lhs}(\mathcal{R}) \,\} + 1$, and

(2) $\sigma(q) \neq \sigma'(p)$ for all $p \in \mathrm{lhs}(\mathcal{R})$ and all substitutions σ and σ'.

For each $p(x_1, \ldots, x_n) \in NI$ and any $(a, H) \in A \times \wp(B)$ with $H = \{b_1, \ldots, b_k\}$, we include in P any rule

$$(a, H)(p(x_1, \ldots, x_n)) \to p[(\bar{a}_1, H_1)(x_1), \ldots, (\bar{a}_n, H_n)(x_n)],$$

where

(a) $p[a_1(x_1), \ldots, a_n(x_n)] \in \mathcal{A}(a, p)$, and

(b) there are terms $s_1 \in \mathcal{B}(b_1, p), \ldots, s_k \in \mathcal{B}(b_k, p)$ such that, for all $i = 1, \ldots, n$,

$$H_i = \mathrm{st}(s_1, x_i) \cup \cdots \cup \mathrm{st}(s_k, x_i).$$

The cases $H = \emptyset$ and $n = 0$ are treated similarly as above.

Type 3. For each $(\bar{a}, H) \in \bar{A} \times \wp(B)$, where $H = \{b_1, \ldots, b_k\}$, we add to P rules as follows.

(1) For $c \in \Sigma_0$, we include in P the rule

$$(\bar{a}, H)(c) \to c$$

if and only if $a(c) \to c$ is in P_1 and P_2 contains $b_i(c) \to c$ for every $b_i \in H$.

(2) For $f \in \Sigma_m$, $m > 0$, we add to P all rules

$$(\bar{a}, H)(f(x_1, \ldots, x_m)) \to f((\bar{a}_1, H_1)(x_1), \ldots, (\bar{a}_m, H_m)(x_m))$$

where

(a) $a(f(x_1, \ldots, x_m)) \to f(a_1(x_1), \ldots, a_m(x_m))$ is in P_1, and

(b) there are rules $b_i(f(x_1, \ldots, x_m)) \to f(b_{i1}(x_1), \ldots, b_{im}(x_m))$ ($i = 1, \ldots, k$) such that for each $j = 1, \ldots, m$, $H_j = \{b_{1j}, \ldots, b_{kj}\}$.

We can show that $\mathcal{C}$ has the property described in $(*)$. If $t \in T(\mathcal{C})$, then $(a_0, \{b_0\})(t) \Rightarrow_{\mathcal{C}}^* t$ and this derivation can be split into two parts

$$(a_0, \{b_0\})(t) \Rightarrow_{\mathcal{C}}^* \tilde{t}[(a_1, H_1)(t_1), \ldots, (a_n, H_n)(t_n)] \Rightarrow_{\mathcal{C}}^* \tilde{t}[t_1, \ldots, t_n] = t, \qquad (**)$$

where $n \geq 0$, $t \in \widetilde{T}_{\Sigma,n}$ and, for every $1 \leq i \leq n$, $t_i \in T_\Sigma$ and $(a_i, H_i) \in A \times \wp(B)$. In the first part of (**) only Type 1 rules are used, and hence $\tilde{t}[a_1(x_1), \ldots, a_n(x_n)] \in \mathcal{A}(a_0, \tilde{t})$. Moreover, for some $k \geq 0$, $\tilde{s} \in \widetilde{T}_{\Sigma,k}$, and $s_1, \ldots, s_k \in T_\Sigma$,

$$\#(t) = \#(\tilde{t}[t_1, \ldots, t_n]) \Rightarrow_{\mathcal{R}_\#} \cdots \Rightarrow_{\mathcal{R}_\#} \tilde{s}[\#(s_1), \ldots, \#(s_k)] = s,$$

where every s_j is a copy of exactly one of the t_i. (Of course, s_j may be equal to more than one t_i.) For each $i = 1, \ldots, n$, let $K(i) = \{\, j \mid s_j \text{ is a copy of } t_i \,\}$. Then for some $u \in \mathcal{B}(b_0, \tilde{s})$, $H_i = \bigcup\{\, \mathrm{st}(u, x_j) \mid j \in K(i) \,\}$ for all $i = 1, \ldots, n$.

In the second part of (**), it is first checked using Type 2 rules that $\tilde{s}[s_1, \ldots, s_k] \in 1r\mathrm{N}_\mathcal{R}(t)$, and the computations $(a_i, H_i)(t_i) \Rightarrow_\mathcal{C}^* t_i$ are finished using Type 3 rules. That means for every $i = 1, \ldots, n$, that (a) $t_i \in T(\mathcal{A}, a_i)$ and (b) $t_i \in T(\mathcal{B}, b)$ for all $b \in H_i$. Therefore

$$a_0(t) \Rightarrow_\mathcal{A}^* \tilde{t}[a_1(t_1), \ldots, a_n(t_n)] \Rightarrow_\mathcal{A}^* \tilde{t}[t_1, \ldots, t_n] = t$$

and there are $b_1, \ldots, b_k \in B$ such that

$$b_0(\tilde{s}[s_1, \ldots, s_k]) \Rightarrow_\mathcal{B}^* \tilde{s}[b_1(s_1), \ldots, b_k(s_k)] \Rightarrow_\mathcal{B}^* \tilde{s}[s_1, \ldots, s_k].$$

The converse of (*) can be proved similarly. □

The corresponding result for sentential forms can be proved by modifying suitably the definition of the recognizer $\mathcal{C}$.

Theorem 4.2. For any left-linear TRS $\mathcal{R} = (\Sigma, R)$, the following *one-pass root-started sentential form inclusion problem* is decidable.

Instance: Recognizable Σ-tree languages T_1 and T_2.

Question: $1r\mathrm{S}_\mathcal{R}(T_1) \subseteq T_2$? □

Proof. Consider a left-linear TRS $\mathcal{R} = (\Sigma, R)$ and any recognizable Σ-tree languages T_1 and T_2. Let $\mathcal{A} = (A, \Sigma, P_1, a_0)$ and $\mathcal{B} = (B, \Sigma, P_2, b_0)$ be top-down Σ-recognizers for which $T(\mathcal{A}) = T_1$ and $T(\mathcal{B}) = T_2^c$. We shall now construct a generalized top-down Σ-recognizer $\mathcal{D}$ such that for any Σ-term t,

$$t \in T(\mathcal{D}) \quad \text{iff} \quad t \in T(\mathcal{A}) \text{ and } s \in T(\mathcal{B}) \text{ for some } s \in 1r\mathrm{S}_\mathcal{R}(t).$$

Then $1r\mathrm{S}_\mathcal{R}(T_1) \subseteq T_2$ if and only if $T(\mathcal{D}) = \emptyset$, and the latter condition is decidable.

Let $\mathcal{D} = (C, \Sigma, P', (a_0, \{b_0\}))$ be the generalized top-down Σ-recognizer, where the state set is that of the recognizer $\mathcal{C}$ used in the proof of Theorem 4.1 and the set P' of transition rules is defined as follows. All rules of $\mathcal{C}$ of Type 1 or of Type 3 will be included also in P'. The rules of Type 2 are replaced by the rules $(a, H)(c) \to c$ and

$$(a, H)(f(x_1, \ldots, x_m)) \to f((\bar{a}_1, H_1)(x_1), \ldots, (\bar{a}_m, H_m)(x_m))$$

which are identical to the Type 3 rules of $\mathcal{C}$ except that the a's have no bars in the left-hand sides.

The recognizer $\mathcal{D}$ is almost the same as $\mathcal{C}$, but it may stop following the chosen one-pass root-started reduction of $\mathcal{R}$ at any moment and switch to states in which

it checks whether the input tree is in $T(\mathcal{A})$ and whether the one-pass root-started sentential form produced by the corresponding simulated reduction sequence of $\mathcal{R}$ is in $T(\mathcal{B})$. □

5 The one-pass leaf-started inclusion problems

First we consider the one-pass leaf-started sentential form inclusion problem. Again the tree languages are assumed to be given in the form of tree recognizers.

Theorem 5.1. For any left-linear TRS $\mathcal{R} = (\Sigma, R)$, the following *one-pass leaf-started sentential form inclusion problem* is decidable.
Instance: Recognizable Σ-tree languages T_1 and T_2.
Question: $1\ell S_{\mathcal{R}}(T_1) \subseteq T_2$?

Proof. Let $\mathcal{A} = (A, \Sigma, P_1, A_f)$ and $\mathcal{B} = (B, \Sigma, P_2, B_f)$ be bottom-up Σ-recognizers that recognize the tree languages T_1 and T_2, respectively. We may assume that $\mathcal{B}$ is total deterministic. We construct a generalized bottom-up Σ-recognizer $\mathcal{C}$ such that $T(\mathcal{C}) = \emptyset$ if and only if $1\ell S_{\mathcal{R}}(T_1) \subseteq T_2$. This recognizer $\mathcal{C} = (C, \Sigma, P, C_f)$ is defined as follows.

(1) Let $C = (A \times B) \cup (\bar{A} \times \bar{B})$, where $\bar{A} = \{ \bar{a} \mid a \in A \}$ and $\bar{B} = \{ \bar{b} \mid b \in B \}$.

(2) The set P consists of the following rules which are of three different types.

Type 1. For every rule $p \to r \in R_e$ with $p, r \in T_{\Sigma,n}$, $n \geq 0$, and for all states $a_1, \ldots, a_n$, $a \in A$, $b_1, \ldots, b_n$, $b \in B$ such that $p[a_1, \ldots, a_n] \Rightarrow_{\mathcal{A}}^* a$ and $r[b_1, \ldots, b_n] \Rightarrow_{\mathcal{B}}^* b$, let P contain the rule $p[(a_1, b_1), \ldots, (a_n, b_n)] \to (a, b)$.

Type 2. For all $a \in A$ and $b \in B$, let $(a, b) \to (\bar{a}, \bar{b})$ be in P.

Type 3. For all $f \in \Sigma_m$ with $m \geq 0$, all $a_1, \ldots, a_m$, $a \in A$ and $b_1, \ldots, b_m$, $b \in B$ such that $f(a_1, \ldots, a_m) \to a \in P_1$ and $f(b_1, \ldots, b_m) \to b \in P_2$, let P contain the rule $f((\bar{a}_1, \bar{b}_1), \ldots, (\bar{a}_m, \bar{b}_m)) \to (\bar{a}, \bar{b})$.

(3) Let $C_f = \{ \bar{a} \mid a \in A_f \} \times \{ \bar{b} \mid b \in (B \setminus B_f) \}$.

The way $\mathcal{C}$ processes a Σ-tree t can be described as follows. First $\mathcal{C}$, using rules of Type 1, follows some one-pass leaf-started rewriting sequences by $\mathcal{R}$ on subtrees of t computing in the first components of its states the evaluations by $\mathcal{A}$ of these subtrees and in the second components the evaluations by $\mathcal{B}$ of the translations of the subtrees produced by these one-pass leaf-started rewriting sequences. At any time $\mathcal{C}$ may switch by rules of Type 2 to a mode in which it by rules of Type 3 computes in the first components of its states the evaluation of t by $\mathcal{A}$ and in the second components the evaluation by $\mathcal{B}$ of the one-pass leaf-started sentential form of t produced by $\mathcal{R}$ when the rewriting sequences on the subtrees are combined. This means that for any $t \in T_\Sigma$, $a \in A$ and $b \in B$,

$$t \Rightarrow_{\mathcal{C}}^* (\bar{a}, \bar{b}) \quad \text{iff} \quad t \Rightarrow_{\mathcal{A}}^* a \text{ and } s \Rightarrow_{\mathcal{B}}^* b \text{ for some } s \in 1\ell S_{\mathcal{R}}(t).$$

By recalling the definition of C_f we see that the above equivalence implies immediately that $T(C) = \emptyset$ if and only if $1\ell S_{\mathcal{R}}(T_1) \subseteq T_2$, as required. □

Finally we consider the one-pass leaf-started normal form inclusion problem.

Theorem 5.2. For any left-linear TRS $\mathcal{R} = (\Sigma, R)$, the following *one-pass leaf-started normal form inclusion problem* is decidable.

Instance: Recognizable Σ-tree languages T_1 and T_2.

Question: $1\ell N_{\mathcal{R}}(T_1) \subseteq T_2$?

Proof. Let $\mathcal{A} = (A, \Sigma, P_1, A_f)$ and $\mathcal{B} = (B, \Sigma, P_2, B_f)$ be total deterministic bottom-up Σ-recognizers such that $T(\mathcal{A}) = T_1$ and $T(\mathcal{B}) = T_2$. We shall construct a generalized bottom-up Σ-recognizer C such that $T(C) = \emptyset$ if and only if $1\ell N_{\mathcal{R}}(T_1) \subseteq T_2$.

Let $mx = \max\{\, \mathrm{hg}(p) \mid p \in \mathrm{lhs}(\mathcal{R}_e) \,\}$ and let $T_{mx} = \{\, t \in \widetilde{T}_{\Sigma,X} \mid \mathrm{hg}(t) \leq mx \,\}$.

Now we define $C = (C, \Sigma, P, C_f)$ as follows.

(1) $C = (A \times B) \cup (\bar{A} \times \bar{B} \times (T_{mx} \cup \{ok\}))$, where $\bar{A} = \{\, \bar{a} \mid a \in A \,\}$ and $\bar{B} = \{\, \bar{b} \mid b \in B \,\}$.

(2) P consists of the following rules which are of five different types.

Type 1. For every rule $p \to r \in R_e$ with $p, r \in T_{\Sigma,n}$, and any states $a_1, \ldots, a_n$, $a \in A$, $b_1, \ldots, b_n$, $b \in B$ such that $p[a_1, \ldots, a_n] \Rightarrow_{\mathcal{A}}^* a$ and $r[b_1, \ldots, b_n] \Rightarrow_{\mathcal{B}}^* b$, let P contain the rule $p[(a_1, b_1), \ldots, (a_n, b_n)] \to (a, b)$.

Type 2. For all $a \in A$ and $b \in B$, let $(a, b) \to (\bar{a}, \bar{b}, x_1)$ be in P.

Type 3. For any $f \in \Sigma_m$ with $m \geq 0$, $a_1, \ldots, a_m$, $a \in A$, $b_1, \ldots, b_m$, $b \in B$ and $u_1, \ldots, u_m \in T_{mx}$ such that $f(a_1, \ldots, a_m) \to a \in P_1$, $f(b_1, \ldots, b_m) \to b \in P_2$, $u = \|f(u_1, \ldots, u_m)\|$ and $u \in T_{mx} \setminus \mathrm{lhs}(\mathcal{R}_e)$, let P contain the rule $f((\bar{a}_1, \bar{b}_1, u_1), \ldots, (\bar{a}_m, \bar{b}_m, u_m)) \to (\bar{a}, \bar{b}, u)$.

In case $m = 0$, the rule has the form $f \to (\bar{a}, \bar{b}, f)$.

Type 4. For any $f \in \Sigma_m$ with $m \geq 0$, $a_1, \ldots, a_m$, $a \in A$, $b_1, \ldots, b_m$, $b \in B$ and $u_1, \ldots, u_m \in T_{mx}$ such that $f(a_1, \ldots, a_m) \to a \in P_1$, $f(b_1, \ldots, b_m) \to b \in P_2$ and $\|f(u_1, \ldots, u_m)\| \notin T_{mx}$, let P contain the rule $f((\bar{a}_1, \bar{b}_1, u_1), \ldots, (\bar{a}_m, \bar{b}_m, u_m)) \to (\bar{a}, \bar{b}, ok)$.

Type 5. For any $f \in \Sigma_m$ with $m \geq 1$, $a_1, \ldots, a_m$, $a \in A$, $b_1, \ldots, b_m$, $b \in B$, and sequence $y_1, \ldots, y_m \in T_{mx} \cup \{ok\}$ such that $ok \in \{y_1, \ldots, y_m\}$, $f(a_1, \ldots, a_m) \to a \in P_1$, $f(b_1, \ldots, b_m) \to b \in P_2$, let P contain the rule $f((\bar{a}_1, \bar{b}_1, y_1), \ldots, (\bar{a}_m, \bar{b}_m, y_m)) \to (\bar{a}, \bar{b}, ok)$.

(3) Let $C_f = \{\, \bar{a} \mid a \in A_f \,\} \times \{\, \bar{b} \mid b \in (B \setminus B_f) \,\} \times (T_{mx} \cup \{ok\})$.

The recognizer C evaluates an input tree t by rules of Type 1 so that it simulates both the computation by $\mathcal{A}$ on subtrees of t and the computation by $\mathcal{B}$ on the translations of those subtrees produced by $\mathcal{R}$. Then C checks that the sentential form produced from t by the computation with $\mathcal{R}$ is a normal form. For this C

switches by rules of Type 2 into a mode in which it, by rules of Type 3, forms in the third components of its states sufficiently large portions of the tree above the nodes where the rewriting with $\mathcal{R}$ ended. If one of these parts turns out to be the left-hand side of a rule of $\mathcal{R}$, then the evaluation by $\mathcal{C}$ stops and t is rejected. If not, then the sentential form produced by $\mathcal{R}$ is a normal form, which is acknowledged by rules of Type 4 by putting *ok* in the third components of the corresponding states of $\mathcal{C}$. Then $\mathcal{C}$ evaluates t in the same way as it was done in the proof of Theorem 5.1 by rules of Type 3.

This means that for any $t \in T_\Sigma$, $a \in A$, $b \in B$, and $y \in T_{mx} \cup \{ok\}$,

$$t \Rightarrow^*_{\mathcal{C}} (\bar{a}, \bar{b}, y) \quad \text{iff} \quad t \Rightarrow^*_{\mathcal{A}} a \text{ and } s \Rightarrow^*_{\mathcal{B}} b \text{ for some } s \in 1\ell\,\mathrm{N}_{\mathcal{R}}(t).$$

Thus $T(\mathcal{C}) = \emptyset$ if and only if $1\ell\,\mathrm{N}_{\mathcal{R}}(T_1) \subseteq T_2$. □

References

[Ave95] J. Avenhaus. *Reduktionssysteme.* Springer, 1995.

[DDC87] M. Dauchet and F. De Comite. A gap between linear and non-linear term-rewriting systems. In *Rewriting Techniques and Applications*, (Proc. Conf., Bordeaux, France, May 25–27, 1987), Lect. Notes Comput. Sci. **256**. Springer, 1987, 95–104.

[DG89] A. Deruyver and R. Gilleron. The reachability problem for ground TRS and some extensions. In *TAPSOFT'89*, (Proc. Conf. CAAP'89, Barcelona, Spain, March 1989), Lect. Notes Comput. Sci. **351**. Springer, 1989, 227–243.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Rewrite Systems*, volume B of *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320. Elsevier, 1990.

[ES77] J. Engelfriet and E. M. Schmidt. IO and OI. I. *J. Comput. Syst. Sci.*, 15(3):328–353, 1977.

[ES78] J. Engelfriet and E. M. Schmidt. IO and OI. II. *J. Comput. Syst. Sci.*, 16(1):67–99, 1978.

[FJSV98] Z. Fülöp, E. Jurvanen, M. Steinby, and S. Vágvölgyi. On one-pass term rewriting. In *MFCS'98*, (Proc. Conf., Brno, Czech Republic, August 1998), Lect. Notes Comput. Sci. **1450**. Springer, 1998, 248–256.

[Gil91] R. Gilleron. Decision problems for term rewriting systems and recognizable tree languages. In *STACS'91*, (Proc. Conf., Hamburg, Germany, February 14–16, 1991), Lect. Notes Comput. Sci. **480**. Springer, 1991, 148–159.

[GS84] F. Gécseg and M. Steinby. *Tree automata.* Akadémiai Kiadó, Budapest, 1984.

[GS97] F. Gécseg and M. Steinby. *Tree Languages*, volume 3 of *Handbook of Formal Languages*, chapter 1, pages 1–68. Springer, 1997.

[GT95] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundam. Inf.*, 24(1,2):157–175, 1995.

[GV98] P. Gyenizse and S. Vágvölgyi. Linear generalized semi-monadic rewrite systems effectively preserve recognizability. *Theor. Comput. Sci.*, 194(1–2):87–122, 1998.

[HH94] D. Hofbauer and M. Huber. Linearizing term rewriting systems using test sets. *J. Symb. Comput.*, 17(1):91–129, 1994.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980.

[KT95] G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. *Inf. Comput.*, 118(1):91–100, 1995.

[VG92] S. Vágvölgyi and R. Gilleron. For a rewrite system it is decidable whether the set of irreducible, ground terms is recognizable. *Bull. EATCS*, 48:197–209, 1992.

Acta Cybernetica 14 (1999) 99-104.

A note on the star-product*

Balázs Imreh † Masami Ito ‡

Dedicated to Professor Ferenc Gécseg on the occasion of his 60th birthday

Abstract

In this paper, we compare the repesenting power of the star-product and the members of two product hierarchies, namely, the α_i-products, $i = 0, 1, \ldots$ and the ν_j-products, $j = 1, 2, \ldots$. In particular, it is proved that the star-product is not isomorphically (homomorphically) more general than any member of this two product families.

The family of the α_i-products, $i = 0, 1, \ldots$, is introduced by F. Gécseg in [3], and a systematic summarizing of the results concerning this product family can be found in the monography [4]. Another product family, the ν_j-products, $j = 1, 2, \ldots$ appears in [2]. Finally, a further product called star-product is studied by M. Tchuente in [8]. The comparison of the representing power of different compositions was initiated by F. Gécseg in [3]. Here, following the idea suggested by him, we compare the representing power of the star-product and the members of the two product families for both the isomorphic and homomorphic representations, and it is shown that the star-product is not isomorphically (homomorphically) more general than the members of the considered families. The inverse problems remain open.

The paper is organized as follows. First, we recall the basic notions and notation. Then, we compare the star-product with the α_i-products, finally, we compare the star-product with the ν_j-products.

By an *automaton* we mean a triplet $\mathbf{A} = (A, X, \delta)$ where A and X are finite nonempty sets, the set of the *states* and the set of the *input symbols*, respectively, and $\delta : A \times X \to A$ is the *transition function*. An automaton $\mathbf{A}$ can be also defined as an algebra $\mathbf{A} = (A, X)$ in which each input symbol is realized as the unary operation $x^{\mathbf{A}} : A \to A$, $a \to \delta(a, x)$. Using the latter definition, the notions such as *subautomaton*, *isomorphism*, and *homomorphism* can be defined in the usual way.

*This work has been supported by the Hungarian National Foundation for Scientific Research, Grant T 014888, the Ministry of Culture and Education of Hungary, Grant FKFP 0704/1997, the Japanese Ministry of Education, Mombusho International Scientific Research Program, Joint Research 10044098

†Department of Informatics, József Attila University, Árpád tér 2, H-6720 Szeged, Hungary

‡Department of Mathematics, Faculty of Science, Kyoto Sangyo University, Kyoto 603, Japan

To define the α_i-product, let $i \geq 0$ be an arbitrarily fixed integer. Let $\mathbf{A}_t = (A_t, X_t)$, $t = 1, \ldots, k$, be automata. Take a nonempty finite set X and a family of mappings $\varphi_t : A_1 \times \cdots \times A_{t+i-1} \times X \to X_t$, $t = 1, \ldots, k$. By the *α_i-product* $\mathbf{B} = \prod_{t=1}^{k} \mathbf{A}_t(X, \varphi)$ we mean the automata $(\prod_{t=1}^{k} A_t, X)$ defined by

$$(a_1, \ldots, a_k)x^{\mathbf{B}} = (a_1 x_1^{\mathbf{A}_1}, \ldots, a_k x_k^{\mathbf{A}_k})$$

where $x_t = \varphi_t(a_1, \ldots, a_{t+i-1}, x)$, $t = 1, \ldots, k$, for all $(a_1, \ldots, a_k) \in \prod_{t=1}^{k} A_t$ and $x \in X$. We note that φ_1 has the form $\varphi_1 : X \to X_1$ when $i = 0$.

A further product family, the ν_j-products, $j = 1, 2, \ldots$, was introduced in [2]. To define this kind of product, let j be an arbitrary positive integer. Then, the *ν_j-product* of automata can be defined in a similar way as the α_i-products, but the family of mappings has the following form:

$$\varphi_t : A_1 \times \cdots \times A_k \times X \to X_t, \quad t = 1, \ldots, k,$$

where each mapping depends on at most j state variables.

A summarizing on the comparisons of the members of the two product families under both isomorphic and homomorphic representations can be found in [1].

To define the star-product, let the automata $\mathbf{A}_t = (A_t, X_t)$, $t = 1, \ldots, k$, be given. Take a nonemty finite set X and a family of mappings $\varphi_1 : A_1 \times \cdots \times A_k \times X \to X_1$ and $\varphi_t : A_1 \times A_t \times X \to X_t$, $t = 2, \ldots, k$. By a *star-product* $\mathbf{A} = (A, X) = \prod_{t=1}^{k} \mathbf{A}_t(X, \varphi)$ (see *e.g.* [6] or [8]) we mean the automaton $(\prod_{t=1}^{k} A_t, X)$ where

$$(a_1, \ldots, a_k)x^{\mathbf{A}} = (ax_1^{\mathbf{A}_1}, \ldots, a_k x_k^{\mathbf{A}_k})$$

with $x_1 = \varphi_1(a_1, \ldots, a_k, x)$ and $x_t = \varphi_t(a_1, a_t, x)$, $t = 2, \ldots, k$, for all $x \in X$ and $(a_1, \ldots, a_k) \in \prod_{s=1}^{k} A_s$.

Now, let β denote one of the products defined above. If in a β-product each component automaton is equal to a given automaton, then this β-product is called a *β-power*. For an arbitrary set $\mathcal{K}$ of automata, let us denote by

$S_\beta(\mathcal{K})$ the β-products of automata from $\mathcal{K}$,

$H(\mathcal{K})$ the homomorphic images of automata from $\mathcal{K}$,

$I(\mathcal{K})$ the isomorphic images of automata from $\mathcal{K}$,

$S(\mathcal{K})$ the subautoma of automata from $\mathcal{K}$.

The class $\mathcal{K}$ is called *isomorphically (homomorphically) complete with respect to the β-product* if $ISP_\beta(\mathcal{K})$ ($HSP_\beta(\mathcal{K})$) is the class of all automata.

Let γ denote one of the products introduced in this paper and which differs from β. It is said that β is *isomorphically (homomorphically) more general than the γ-product* if $ISP_\gamma(\mathcal{K}) \subseteq ISP_\beta(\mathcal{K})$ ($HSP_\gamma(\mathcal{K}) \subseteq HSP_\beta(\mathcal{K})$) is valid for every set $\mathcal{K}$ of automata, moreover, there exists a set $\mathcal{K}_0$ of automata such that $ISP_\gamma(\mathcal{K}_0) \subset ISP_\beta(\mathcal{K}_0)$ ($HSP_\gamma(\mathcal{K}_0) \subset HSP_\beta(\mathcal{K}_0)$).

Now, we are ready to present our results. Regarding the α_i-products and isomorphic representation, the following statement is valid.

Theorem 1. *The star-product is not isomorphically more general than the α_i-product, for all i, $i = 0, 1 \ldots$.*

Proof. For an arbitrary positive integer n, let us define the automaton $\mathbf{I}_n = (\{1, \ldots, n\}, \{x\})$ by $nx^{\mathbf{I}_n} = n$ and $jx^{\mathbf{I}_n} = j + 1$, $j = 1, \ldots, n-1$. Furthermore, let the automaton $\mathbf{E} = (\{0, 1\}, \{x, y\})$ be defined by $0y^{\mathbf{E}} = 0$, $0x^{\mathbf{E}} = 1x^{\mathbf{E}} = 1y^{\mathbf{E}} = 1$. It is known (*cf.* [4] or [7]) that $\mathcal{K} = \{\mathbf{E}\}$ is isomorphically complete for the class of nilpotent automata with respect to the α_0-product, *i.e.*, every nilpotent automaton can be embedded isomorphically into an α_0-power of $\mathbf{E}$. Hence, $\mathbf{I}_n \in ISP_{\alpha_0}(\{\mathbf{E}\})$, for every positive integer n.

Now, we will show that if $\mathbf{I}_n \in ISP_{\text{star}}(\{\mathbf{E}\})$, then $n \leq 4$. For this purpose, let us suppose that $\mathbf{I}_n$ can be embedded isomorphically into a star-power $\mathbf{A} = \prod_{t=1}^{s} \mathbf{E}_t(\{x\}, \varphi)$ where $\mathbf{E}_t$ denotes the t-th copy of $\mathbf{E}$. Without loss of generality, we may assume that s is minimal with this property, *i.e.*, if $\mathbf{I}_n$ is isomorphic to a subautomaton of a star-power of $\mathbf{E}$, then the number of the factors of the star-power considered is at least s. Let μ be a suitable isomorphism and $\mu(j) = (a_{j1}, \ldots, a_{js})$, $j = 1, \ldots, n$. If $a_{1t} = 1$ for some $t \in \{1, \ldots, s\}$, then $a_{jt} = 1$, $j = 1, \ldots, n$, by the definition of $\mathbf{E}$. Then, $\mathbf{E}_t$ can be omitted from the star-product $\mathbf{A}$ which contradicts the minimality of s. Consequently, we may assume that $a_{1t} = 0$, $t = 1, \ldots, s$. Furthermore, let us observe that if $a_{jr} = a_{jt}$, $j = 1, \ldots, n$, for some integers $r \neq t \in \{1, \ldots, s\}$, then omitting one of the automata $\mathbf{E}_r$ and $\mathbf{E}_t$, the automaton $\mathbf{I}_n$ can be embedded isomorphically into the remaining star-product which contradicts the minimality of s. Therefore, we may assume that the vectors $(a_{1t}, a_{2t}, \ldots, a_{nt})^T$, $t = 1, \ldots, s$, are pairwise different and none of them is equal to the n-dimensional zero vector. Now, let us classify the automata $\mathbf{E}_2, \mathbf{E}_3, \ldots, \mathbf{E}_s$ into the classes M_1 and M_2 dependig on the values of the mappings $\varphi_2, \varphi_3, \ldots, \varphi_s$ as follows:

$$M_1 = \{\mathbf{E}_t : 2 \leq t \leq s \ \ \& \ \ \varphi_t(0, 0, x) = x\},$$

$$M_2 = \{\mathbf{E}_t : 2 \leq t \leq s \ \ \& \ \ \varphi_t(0, 0, x) = y\}.$$

If $\mathbf{E}_t \in M_1$, then let us observe that $(a_{1t}, \ldots, a_{nt})^T = (0, 1, \ldots, 1)$, and hence, the star-product $\mathbf{A}$ may contain at most one element from M_1, by the minimality of s. If $\mathbf{E}_t \in M_2$, then there are two possibilities for $\varphi_t(1, 0, x)$. If $\varphi_t(1, 0, x) = y$, then $(a_{1t}, \ldots, a_{nt})^T = (0, \ldots, 0)$, and therefore, $\varphi_t(1, 0, x) = x$ must hold, by the minimality of s. But in this case, $a_{1t} = a_{2t} = \ldots a_{it} = 0$, $a_{i+1,t} = \ldots = a_{nt} = 1$ for some $i > 2$, provided that $a_{11} = a_{21} = \ldots = a_{i-2,1} = 0$ and $a_{i-1,1} = \ldots = a_{n1} = 1$. Hence, the star-product $\mathbf{A}$ may contain only one automaton from M_2. Consequently, $s \leq 3$. Now, it is easy to see that starting from the 3-dimensional zero vector, $(0, 0, 0)(xx)^{\mathbf{A}} = (1, 1, 1)$ if $a_{21} = 1$ and $(0, 0, 0)(xxx)^{\mathbf{A}} = (1, 1, 1)$ if $a_{21} = 0$ and $a_{31} = 1$, furthermore, $a_{21} = a_{31} = 0$ is impossible. Thus, $n \leq 4$ which yields that $ISP_{\alpha_0}(\{\mathbf{E}\}) \not\subseteq ISP_{\text{star}}(\{\mathbf{E}\})$. On the other hand, $ISP_{\alpha_0}(\{\mathbf{E}\}) \subseteq ISP_{\alpha_i}(\{\mathbf{E}\})$,

for all i, $i = 0, 1, \ldots$, and hence, $ISP_{\alpha_i}(\{\mathbf{E}\}) \not\subseteq ISP_{\text{star}}(\{\mathbf{E}\})$ is valid for every nonnegative integer i. This completes the proof of our statement.

Remark 1. It is an open problem whether there exists a nonnegative integer i such that the α_i-product is isomorphically more general than the star-product.

Regarding the homomorphic representation, the following statement is valid.

Theorem 2. *The star-product is not homomorphically more general than the α_i-product, for all i, $i = 0, 1, \ldots$.*

Proof. For arbitrary integers $k > 0$ and $l \geq 0$, let the automaton $\mathbf{J}_{k,l} = (\{1, \ldots, k, k+1, \ldots, k+l\}, \{x\})$ be defined by $(k+l)x^{\mathbf{J}_{k,l}} = k$ and $jx^{\mathbf{J}_{k,l}} = j+1$, $j = 1, \ldots, k+l-1$. Then, it can be easily seen that $\mathbf{I}_n \in HSP_{\text{star}}(\{\mathbf{E}\})$ if and only if $\mathbf{J}_{k,l} \in ISP_{\text{star}}(\{\mathbf{E}\})$ for some $k \geq n$ and $l \geq 0$. By the proof of Theorem 1, $\mathbf{J}_{k,l} \in ISP_{\text{star}}(\{\mathbf{E}\})$ implies $k \leq 4$. Therefore, $\mathbf{I}_n \in HSP_{\text{star}}(\{\mathbf{E}\})$ yields that $n \leq 4$. On the other hand, $ISP_{\alpha_0}(\{\mathbf{E}\}) \subseteq HSP_{\alpha_0}(\{\mathbf{E}\})$, and thus, $\mathbf{I}_n \in HSP_{\alpha_0}(\{\mathbf{E}\})$ for every positive integer n. Now, we can obtain the validity of Theorem 2 in a similar way as above.

Remark 2. It is an open problem whether there is a nonnegative integer i such that the α_i-product is homomorphically more general than the star-product.

Regarding the ν_j-products, we have the following assertion for the isomorphic representation.

Theorem 3. *The star-product is not isomorphically more general than the ν_j-product, for all j, $j = 1, 2, \ldots$.*

Proof. For an arbitrary positive integer n, let us define the automaton $\mathbf{C}_n = (\{0, 1, \ldots, n-1\}, \{x\})$ by $ix^{\mathbf{C}_n} = i+1 \pmod n$. $\mathbf{C}_n$ is called a *counter of length n*. Furthermore, let $\mathbf{B} = (\{0,1\}, \{x, y\})$ be defined by $0x^{\mathbf{B}} = 1x^{\mathbf{B}} = 1$, $1y^{\mathbf{B}} = 0y^{\mathbf{B}} = 0$. We show that $\mathbf{C}_n \in ISP_{\nu_1}(\{\mathbf{B}\})$ is valid for every positive integer n. For this purpose, let n be an arbitrary positive integer. Let us form the ν_1-power $\mathbf{B}^n(\{x\}, \varphi)$ as follows. For every $(a_1, \ldots, a_n) \in \{0,1\}^n$ and $t \in \{2, \ldots, n\}$, let

$$\varphi_t(a_1, \ldots, a_n, x) = \varphi_t(a_{t-1}, x) = \begin{cases} x & \text{if } a_{t-1} = 1, \\ y & \text{otherwise,} \end{cases}$$

and

$$\varphi_1(a_1, \ldots, a_n, x) = \varphi_1(a_n, x) = \begin{cases} x & \text{if } a_n = 1, \\ y & \text{otherwise.} \end{cases}$$

Then, it is easy to prove that $\mathbf{C}_n$ can be embedded isomorphically into the defined ν_1-power, $\mathbf{B}^n(\{x\}, \varphi)$. For example, μ is an appropriate isomorphism where μ is defined by

$$\mu(0) = (1, 0, 0, \ldots, 0, 0)$$

$$\mu(1) = (0,1,0,\ldots,1,1)$$

$$\vdots \qquad\qquad \vdots$$

$$\mu(n-1) = (0,0,0,\ldots,0,1)$$

Now, we show that if $\mathbf{C}_n \in ISP_{\mathrm{star}}(\{\mathbf{B}\})$, then $n \leq 2^{2^5-1}$. For this purpose, let us suppose that $\mathbf{C}_n$ can be embedded isomorphically into a star-power $\mathbf{A} = \prod_{t=1}^{s} \mathbf{B}_t(\{x\},\varphi)$ where $\mathbf{B}_t$ denotes the t-th copy of $\mathbf{B}$. We may assume that s is minimal with this property. Let μ be a suitable isomorphism and $\mu(i) = (a_{i1},\ldots,a_{is})$, $i = 0,1,\ldots n-1$. Let us classify the automata $\mathbf{B}_2,\mathbf{B}_3,\ldots,\mathbf{B}_s$, into the classes M_{r,z_1,z_2,z_3,z_4}, $r \in \{0,1\}$, $z_l \in \{x,y\}$, $l = 1,\ldots,4$ where

$$M_{r,z_1,z_2,z_3,z_4} = \{\mathbf{B}_t : 2 \leq t \leq s \ \ \& \ \ r = a_{0,t} \ \ \& \ \ \varphi_t(0,0,x) = z_1 \ \ \&$$

$$\varphi_t(0,1,x) = z_2 \ \ \& \ \ \varphi_t(1,0,x) = z_3 \ \ \& \ \ \varphi_t(1,1,x) = z_4\}$$

It is easy to show that if $\mathbf{B}_u$ and $\mathbf{B}_v$ are the same class for some integers $u,v \in \{2,3,\ldots,s\}$, then one of them can be omitted from the star-product considered which contradicts the minimality of s. Consequently, $\mathbf{A}$ may contain at most one factor from each class. Then, $s \leq 2^5 + 1$, and therefore, $n \leq 2^{2^5+1}$. From this it follows that $ISP_{\nu_1}(\{\mathbf{B}\}) \not\subseteq ISP_{\mathrm{star}}(\{\mathbf{B}\})$. On the other hand, by the definition of the ν_j-products, $ISP_{\nu_1}(\{\mathbf{B}\}) \subseteq ISP_{\nu_j}(\{\mathbf{B}\})$, for all j, $j = 1,2,\ldots$, and hence, $ISP_{\nu_j}(\{\mathbf{B}\}) \not\subseteq ISP_{\mathrm{star}}(\{\mathbf{B}\})$ is valid for every positive integer j.

Remark 3. It remains an open problem whether there exists a positive integer j such that the ν_j-product is isomorphically more general than the star-product.

For the homomorphic representation, we can conclude the following assertion.

Theorem 4. *The star-product is not homomorphically more general than the ν_j-product, for all j, $j = 1,2,\ldots$.*

Proof. Since $ISP_{\nu_1}(\{\mathbf{B}\}) \subseteq HSP_{\nu_1}(\{\mathbf{B}\})$, we have $\mathbf{C}_n \in HSP_{\nu_1}(\{\mathbf{B}\})$, for every positive integer n. On the other hand, it is easy to see that $\mathbf{C}_n \in HSP_{\mathrm{star}}(\{\mathbf{B}\})$ if and only if $\mathbf{C}_m \in ISP_{\mathrm{star}}(\{\mathbf{B}\})$ for some multiple m of n. Thus, by the proof of Theorem 3, we obtain that $\mathbf{C}_n \in HSP_{\mathrm{star}}(\{\mathbf{B}\})$ implies $n \leq 2^{2^5+1}$. This results in $HSP_{\nu_1}(\{\mathbf{B}\}) \not\subseteq HSP_{\mathrm{star}}(\{\mathbf{B}\})$, and then, by the definition of the ν_j-product, we obtain the validity of Theorem 4.

Remark 4. It is an open problem whether there is a positive integer j such that the ν_j-product is homomorphically more general than the star-product.

Conjecture. We think so that the problems presented as open ones have negative answers, and thus, the star-product is incomparable with the members of the considered product families with respect to both the isomorphic and homomorphic representations. The proof of this conjecture may need further deep investigations.

References

[1] Dömösi, P., Z. Ésik, B. Imreh, On product hierarchies of automata, Proceedings of the 7th Conference on Fundamentals of Computation Theory, Szeged, LNCS **380**, Springer-Verlag, Berlin, 1989, 137-144.

[2] Dömösi P., B. Imreh, On ν_i-products of automata, *Acta Cybernetica* **7** (1983), 149-162.

[3] Gécseg, F., Composition of automata, Proceedings of the 2nd Colloquium on Automata, Languages and Programming, Saarbrücken, LNCS **14**, Springer-Verlag, Berlin, 1974, 351-363.

[4] Gécseg, F., *Products of automata*, Springer-Verlag, Berlin-Heidelberg-New York-Tokyo, 1986.

[5] Gécseg, F., On ν_i-products of commutative automata, *Acta Cybernetica* **7** (1985), 55-59.

[6] Gécseg, F., B. Imreh, On star-products of automata, *Acta Cybernetica* **9** (1989), 167-172.

[7] Imreh, B., On finite nilpotent automata, *Acta Cybernetica* **5** (1981), 281-239.

[8] Tchuente, M., Computation on finite network of automata, in: C. Choffrut (Ed.), Automata Networks, LNCS **316**, Springer-Verlag, Berlin, 1986, 53-67.

Acta Cybernetica **14** (1999) 105-115.

Directable nondeterministic automata*

B. Imreh † M. Steinby ‡

Dedicated to Professor Ferenc Gécseg on the occasion of his 60th birthday

Abstract

An automaton is directable if it has a directing word which takes it from every state to the same state. For nondeterministic (n.d.) automata directability can be defined in several meaningful ways. We consider three such notions. An input word w of an n.d. automaton $\mathcal{A}$ is

(1) D1-directing if the set of states aw in which $\mathcal{A}$ may be after reading w consists of the same single state c for all initial states a;

(2) D2-directing if the set aw is independent of the initial state a;

(3) D3-directing if some state c appears in all of the sets aw.

We consider the sets of D1-, D2- and D3-directing words of a given n.d. automaton, and compare the classes of D1-, D2- and D3-directable n.d. automata with each other. We also estimate the lengths of the longest possible minimum-length D1-, D2- and D3-directing words of an n-state n.d. automaton. All questions are studied separately for n.d. automata which have at least one next state for every input-state pair.

1 Introduction

An input word w is called a *directing* (or *synchronizing*) *word* of an automaton $\mathcal{A}$ if it takes $\mathcal{A}$ from every state to the same fixed state, *i.e.* if there is a state c such that $aw = c$ for all states a of $\mathcal{A}$. An automaton is *directable* if it has a directing word. Directable automata and directing words have been studied extensively in the literature from various points of view (*cf.* [1],[3], [4],[8],[9], [10],[11], for example). The main challenge from the very beginning has been Černý's Conjecture [3] which claims that any n-state ($n \geq 1$) directable automaton has a directing word of length $(n-1)^2$ or less. The bound suggested by the conjecture is the lowest possible, but the best known upper bounds are of order $\mathcal{O}(n^3)$, and the conjecture remains

*This work has been supported by the Hungarian National Foundation for Scientific Research, Grant T014888, the Hungarian-Finnish S & T Co-operation Programme for 1997-1999, Grant SF-10/97 and the Ministry of Culture and Education of Hungary, Grant PFP-4123/1997.

†Department of Informatics, József Attila University, Árpád tér 2, H-6720 Szeged, Hungary

‡TUCS and Department of Mathematics, University of Turku, FIN-20014 Turku, Finland

unsettled. On the other hand, for some special classes of automata even better and accurate bounds have been found (*cf.* [8],[9],[10],).

For nondeterministic (n.d.) automata directability can be defined in several meaningful ways three of which we will study here. An input word w of an n.d. automaton $\mathcal{A}$ is

(1) D1-directing if the set of states aw in which $\mathcal{A}$ may be after reading w consists of the same single state c whatever the initial state a is;

(2) D2-directing if the set aw is independent of the initial state a;

(3) D3-directing if there exists a state c which appears in all sets aw.

The D1-directability of complete n.d. automata was already studied by Burkhard [1]. In a paper [6] on games of composing relations on a finite set Goralčik *et al.*, in effect, considered D1- and D3-directability and also two special types of D2-directability for general n.d. automata. In [1] an exact exponential bound for the length of minimum-length D1-directing words of complete n.d. automata was given, and in [6] it was shown that neither for D1- nor for D3-directing words the bound can be polynomial for general n.d. automata, and that the same holds for the two special types of D2-directability considered. On the other hand, Carpi [2] has found $\mathcal{O}(n^3)$ bounds for D1-directing words of unambiguous automata and for synchronizing pairs of maximal rational codes recognized by such automata.

In this paper we consider the three types of directability from several points of view both for complete n.d. automata and for general n.d. automata. Section 2 contains the general preliminaries. In Section 3 we give the formal definitions of D1-, D2 and D3-directing words and study the sets of Di-directing words of a given automaton ($i = 1, 2, 3$). Moreover, a diagram showing the inclusion relationships between the various classes of directable n.d. automata is presented. In Section 4 we study the preservation of directability properties when one forms subautomata, epimorphic images or finite direct products of n.d. automata. Finally, in Section 5 we derive lower and upper bounds for the lengths of the shortest directing words of the three different types. For D1-directing words it only remained to note that Burkhard's exact value applies also to general n.d. automata. For D2- and D3-directing words exponential lower bounds are obtained by utilizing an idea used in [6], and by considering recognizers of the sets of D2- or D3-directing words of a given automaton we get also upper bounds for them. The gaps between the lower bounds and the upper bounds are, however, considerable. Here the D3-directing words of complete n.d. automata form a notable exception: for them we obtain a lower bound of order $\mathcal{O}(n^2)$ and an upper bound of order $\mathcal{O}(n^3)$.

2 Preliminaries

The cardinality of a set A is denoted by $|A|$. If $f : A \to B$ is a mapping, the image $f(a)$ of an element $a \in A$ is often denoted by af. Similarly, we may write Hf for $f(H) = \{af : a \in H\}$ when $H \subseteq A$. The composition of two mappings $f : A \to B$

and $g : B \to C$ is the mapping $fg : A \to C$, $a \mapsto (af)g$, and the product of two relations $\theta \subseteq A \times B$ and $\rho \subseteq B \times C$ is the relation

$$\theta\rho = \{(a, c) \in A \times C : (\exists b \in B)\ a\theta b,\ b\rho c\}$$

from A to C; that $(a, b) \in \theta$ holds is also expressed by writing $a\theta b$.

In what follows, X is always a finite nonempty alphabet. The set of all (finite) words over X is denoted by X^* and the empty word by ε. The length of a word w is denoted by $\lg(w)$.

An *automaton* is a system $\mathcal{A} = (A, X, \delta)$, where A is a finite nonempty set of *states*, X is the *input alphabet*, and $\delta : A \times X \to A$ is the *transition function*. The transition function is extended to $A \times X^*$ in the usual way. A *recognizer* is a system $\mathbf{A} = (A, X, \delta, a_0, F)$, where (A, X, δ) is an automaton, $a_0 (\in A)$ is the *initial state*, and $F (\subseteq A)$ is the set of *final states*. The *language recognized* by $\mathbf{A}$ is the set

$$L(\mathbf{A}) = \{w \in X^* : \delta(a_0, w) \in F\}.$$

A language is called *recognizable*, or *regular*, if it is recognized by some recognizer. The set of all recognizable languages over the alphabet X is denoted by $\mathrm{Rec}(X)$.

An automaton $\mathcal{A} = (A, X, \delta)$ can also be defined as an algebra $\mathcal{A} = (A, X)$ in which each input letter x is realized as the unary operation $x^{\mathcal{A}} : A \to A$, $a \mapsto \delta(a, x)$. Nondeterministic automata may then be introduced as generalized automata in which the unary operations are replaced by binary relations. Hence a *nondeterministic (n.d.) automaton* is a system $\mathcal{A} = (A, X)$ where A is a finite nonempty set of *states*, X is the *input alphabet*, and each letter $x (\in X)$ is realized as a binary relation $x^{\mathcal{A}} (\subseteq A \times A)$ on A. For any $a \in A$ and $x \in X$, $ax^{\mathcal{A}} = \{b \in A : (a, b) \in x^{\mathcal{A}}\}$ is the set of states into which $\mathcal{A}$ may enter from state a by reading the input letter x. For any $C \subseteq A$ and $x \in X$, we set $Cx^{\mathcal{A}} = \bigcup\{ax^{\mathcal{A}} : a \in C\}$. For $w \in X^*$ and $C \subseteq A$, $Cw^{\mathcal{A}}$ is obtained inductively thus:

(1) $C\varepsilon^{\mathcal{A}} = C$;

(2) $Cw^{\mathcal{A}} = (Cv^{\mathcal{A}})x^{\mathcal{A}}$ for $w = vx$, $v \in X^*$ and $x \in X$.

If $w \in X^*$ and $a \in A$, let $aw^{\mathcal{A}} = \{a\}w^{\mathcal{A}}$. This means that if $w = x_1 x_2 \dots x_k$, then $w^{\mathcal{A}} = x_1^{\mathcal{A}} x_2^{\mathcal{A}} \dots x_k^{\mathcal{A}} (\subseteq A \times A)$. If C is the set in which $\mathcal{A}$ could be at a certain moment, then by the usual interpretation of nondeterminism $Cw^{\mathcal{A}}$ is the set of possible states after $\mathcal{A}$ has received the input word w. When $\mathcal{A}$ is known from the context, we usually write simply aw and Cw for $aw^{\mathcal{A}}$ and $Cw^{\mathcal{A}}$, respectively.

An n.d. automaton $\mathcal{A} = (A, X)$ is *complete* if $ax^{\mathcal{A}} \neq \emptyset$ for all $a \in A$ and $x \in X$. Complete n.d. automata are called c.n.d. automata for short. In what follows, we denote a deterministic automaton by $\mathcal{A} = (A, X, \delta)$ and a nondeterministic automaton by $\mathcal{A} = (A, X)$.

3 Directable nondeterministic automata

An automaton $\mathcal{A} = (A, X, \delta)$ is said to be *directable* if it has a *directing word* $w(\in X^*)$ such that $\delta(a, w) = \delta(b, w)$ for all $a, b \in A$ (cf. [3],[4], [8],[11], for example). Hence a directing word sends the automaton to a known state which is independent of the present state. This idea can be extended to n.d. automata in several nonequivalent ways. In the following definition three natural notions of directability of n.d. automata are introduced.

Definition 3.1. Let $\mathcal{A} = (A, X)$ be an n.d. automaton. For any word $w \in X^*$ we consider the following three conditions:

(D1) $(\exists c \in A)(\forall a \in A)(aw = \{c\})$;

(D2) $(\forall a, b \in A)(aw = bw)$;

(D3) $(\exists c \in A)(\forall a \in A)(c \in aw)$.

If w satisfies (Di), then w is a Di-*directing word* of $\mathcal{A}$ $(i = 1, 2, 3)$. For each $i = 1, 2, 3$, the set of Di-directing words of $\mathcal{A}$ is denoted by $\mathrm{D}_i(\mathcal{A})$, and $\mathcal{A}$ is called Di-*directable* if $\mathrm{D}_i(\mathcal{A}) \neq \emptyset$. The classes of D$i$-directable n.d. automata and c.n.d. automata are denoted by $\mathbf{Dir}(i)$ and $\mathbf{CDir}(i)$, respectively.

A D1-directing word drives the n.d. automaton from any state, or any nonempty set of states, to some fixed state. D2-directability generalizes the notion of directability in the sense that after reading a D2-directing word the set of possible states is independent of the starting state. In fact, if $w \in \mathrm{D}_2(\mathcal{A})$, then the set $Cw^{\mathcal{A}}$ is independent of $C(\subseteq A)$ as long as $C \neq \emptyset$. Finally, if w is a D3-directing word of $\mathcal{A} = (A, X)$, then at least one state in $Cw^{\mathcal{A}}$ is known even if the initial set C of possible states is unknown, but nonempty. Of course, if the n.d. automaton is complete, the current set of possible states is always nonempty.

In [1] Burkhard considered D1-directing words ("homogeneous experiments") for complete n.d. automata. The game of composing a constant relation from a set of relations studied in [6] amounts to the forming of a D1-directing word for an n.d. automaton, and other variants of such games correspond our D3-directability and two special types of D2-directability.

Let us begin the study of the relationships between the various notions of directibility by considering the directing words of a given n.d. automaton $\mathcal{A} = (A, X)$. For this purpose and future use we define the n.d. automata $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ by the following transition tables:

$\mathcal{A}_1$	x
1	2
2	1, 2

$\mathcal{A}_2$	x	y
1	2	–
2	1	–

$\mathcal{A}_3$	x
1	1, 2
2	2

$\mathcal{A}_4$	x	y
1	1, 2	–
2	2	–

It is clear that any D1-directing word is also D2- and D3-directing. Moreover, the inclusion $\mathrm{D}_1(\mathcal{A}_1) \subseteq \mathrm{D}_2(\mathcal{A}_1) \cap \mathrm{D}_3(\mathcal{A}_1)$ is proper since the word xx is D2- and D3-directing, but not D1-directing. That $\mathrm{D}_2(\mathcal{A})$ and $\mathrm{D}_3(\mathcal{A})$ may be incomparable can

be seen by considering the n.d. automaton $\mathcal{A}_4$; obviously $x \in \mathrm{D}_3(\mathcal{A}_4) \setminus \mathrm{D}_2(\mathcal{A}_4)$ and $y \in \mathrm{D}_2(\mathcal{A}_4) \setminus \mathrm{D}_3(\mathcal{A}_4)$. On the other hand, it is clear that $\mathrm{D}_1(\mathcal{A}) \subseteq \mathrm{D}_2(\mathcal{A}) \subseteq \mathrm{D}_3(\mathcal{A})$ for any c.n.d. automaton $\mathcal{A}$. That both of these inclusions may be proper can be seen by considering the c.n.d. automaton $\mathcal{A}_1$. These observations may be summed up as follows.

Remark 3.2. For any n.d. automaton $\mathcal{A}$, $\mathrm{D}_1(\mathcal{A}) \subseteq \mathrm{D}_2(\mathcal{A}) \cap \mathrm{D}_3(\mathcal{A})$, and if $\mathcal{A}$ is complete, then $\mathrm{D}_1(\mathcal{A}) \subseteq \mathrm{D}_2(\mathcal{A}) \subseteq \mathrm{D}_3(\mathcal{A})$. Moreover, any one of the inclusions may be proper.

The following observations are also easily verified.

Remark 3.3. For any n.d. automaton $\mathcal{A} = (A, X)$, $\mathrm{D}_2(\mathcal{A})X^* = \mathrm{D}_2(\mathcal{A})$. If $\mathcal{A}$ is complete, then $X^*\mathrm{D}_1(\mathcal{A}) = \mathrm{D}_1(\mathcal{A})$, $X^*\mathrm{D}_2(\mathcal{A})X^* = \mathrm{D}_2(\mathcal{A})$, and $X^*\mathrm{D}_3(\mathcal{A})X^* = \mathrm{D}_3(\mathcal{A})$.

Next we note that the directing words of each type of any given n.d. automaton form a regular language.

Proposition 3.4. *For any n.d. automaton $\mathcal{A}$, the languages $\mathrm{D}_1(\mathcal{A})$, $\mathrm{D}_2(\mathcal{A})$ and $\mathrm{D}_3(\mathcal{A})$ are (effectively) recognizable.*

Proof. Let $\mathcal{A} = (A, X)$ be any n.d. automaton. Suppose that $\mathcal{A}$ has n states and let $A = \{a_1, \ldots, a_n\}$. First we define an automaton $\mathcal{B} = (B, X, \delta)$ so that $B = \{\{a_1u^{\mathcal{A}}, \ldots, a_nu^{\mathcal{A}}\} : u \in X^*\}$ and $\delta(\{C_1, \ldots, C_k\}, x) = \{C_1x^{\mathcal{A}}, \ldots, C_kx^{\mathcal{A}}\}$ for all $\{C_1, \ldots, C_k\} \in B$ and $x \in X$. Furthermore, let $b_0 = \{\{a_1\}, \ldots, \{a_n\}\} (\in B)$. It is clear that $\delta(b_0, u) = \{a_1u^{\mathcal{A}}, \ldots, a_nu^{\mathcal{A}}\}$ for every $u \in X^*$. Hence $L(\mathbf{B}_i) = \mathrm{D}_i(\mathcal{A})$ for $\mathbf{B}_i = (B, X, \delta, b_0, F_i)$, $i = 1, 2, 3$, when we set $F_1 = \{\{\{c\}\} : c \in A\} \cap B$, $F_2 = \{\{C\} : C \subseteq A\} \cap B$ and $F_3 = \{\{C_1, \ldots, C_k\} : C_1 \cap \ldots \cap C_k \neq \emptyset\} \cap B$. The constructions of the recognizers $\mathbf{B}_1$, $\mathbf{B}_2$ and $\mathbf{B}_3$ are clearly effective.

Corollary 3.5. The D1-, D2- and D3-directability of an n.d. automaton are decidable properties.

Next we investigate the relationships between the various classes $\mathbf{Dir}(i)$ and $\mathbf{CDir}(i)$.

Proposition 3.6 *The pairwise inclusion relations between the classes $\mathbf{Dir}(i)$ and $\mathbf{CDir}(i)$, $i = 1, 2, 3$, are given by the Hasse diagram shown in Figure 1. All inclusions are proper and the pairwise intersections are as indicated by the diagram.*

Proof. Since $\mathcal{A}_2 \in \mathbf{Dir}(2) \setminus \mathbf{Dir}(3)$ and $\mathcal{A}_3 \in \mathbf{Dir}(3) \setminus \mathbf{Dir}(2)$, the classes $\mathbf{Dir}(2)$ and $\mathbf{Dir}(3)$ are incomparable and $\mathbf{Dir}(2) \cap \mathbf{Dir}(3)$ is contained properly in both of them. The inclusion $\mathbf{Dir}(1) \subseteq \mathbf{Dir}(2) \cap \mathbf{Dir}(3)$ follows from Remark 3.2, and its properness is witnessed by $\mathcal{A}_1$. The inclusions $\mathbf{CDir}(1) \subseteq \mathbf{CDir}(2) \subseteq \mathbf{CDir}(3)$, also implied by Remark 3.2, are proper since $\mathcal{A}_1 \in \mathbf{CDir}(2) \setminus \mathbf{CDir}(1)$ and $\mathcal{A}_3 \in \mathbf{CDir}(3) \setminus \mathbf{CDir}(2)$. It is clear that $\mathbf{CDir}(i) \subset \mathbf{Dir}(i)$ for every $i = 1, 2, 3$, and it follows now directly from the definitions that the intersections of all pairs of classes are correctly given by the diagram.

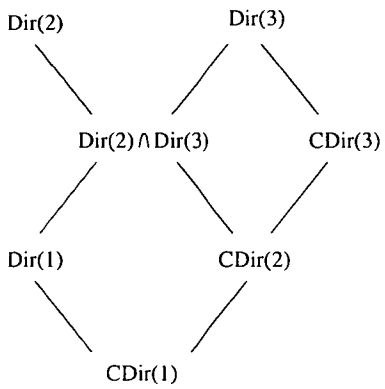


Figure 1:

4 Algebraic constructions and directability

In [8] it was noted that subautomata, epimorphic images and finite direct products of directable automata are directable. Here we consider these matters for nondeterministic automata. Throughout this section $\mathcal{A} = (A, X)$ and $\mathcal{B} = (B, X)$ are n.d. automata which have the same input alphabet.

Let us call $\mathcal{B}$ a *subautomaton* of $\mathcal{A}$ if $B \subseteq A$ and $bx^{\mathcal{B}} = bx^{\mathcal{A}}$ for all $b \in B$ and $x \in X$. It is easy to show that if $\mathcal{B}$ is a subautomaton of $\mathcal{A}$, then $bw^{\mathcal{B}} = bw^{\mathcal{A}}$ for all $b \in B$ and $w \in X^*$. This observation yields immediately the following facts.

Proposition 4.1. *If $\mathcal{B}$ is a subautomaton of an n.d. automaton $\mathcal{A}$, then $\mathrm{D}_i(\mathcal{A}) \subseteq \mathrm{D}_i(\mathcal{B})$, and hence every subautomaton of a $\mathrm{D}i$-directable n.d. automaton is $\mathrm{D}i$-directable, $i = 1, 2, 3$.*

In [5] a weaker notion of subautomaton was used which can be derived from the general notion of a substructure (cf. [7], for example). Let us say that $\mathcal{B}$ is a *weak subautomaton* of $\mathcal{A}$ if $B \subseteq A$ and $bx^{\mathcal{B}} = bx^{\mathcal{A}} \cap B$ for all $b \in B$ and $x \in X$. None of the claims of Proposition 4.1 holds for weak subautomata.

Example 4.2. The n.d. automaton $\mathcal{A} = (\{1,2,3\}, \{x\})$, where $x^{\mathcal{A}} = \{(1,2),(2,3),(3,3)\}$ is D1-, D2- and D3-directable, but the weak subautomaton corresponding to the subset $\{1,3\}$ has none of these properties. The c.n.d. automaton $\mathcal{B} = (\{1,2,3\}, \{x\})$, where $x^{\mathcal{B}} = \{(1,2),(2,1),(2,3),(3,2),(3,3)\}$ is D2- and D3-directable, but the weak subautomaton $(\{1,2\}, \{(1,2),(2,1)\})$ is neither D2- nor D3-directable although it is complete.

Also homomorphisms of n.d. automata can be defined in different ways. We consider a notion used in [5]: a mapping $\varphi : A \to B$ is a *morphism* from $\mathcal{A}$ to $\mathcal{B}$, and we express this by writing $\varphi : \mathcal{A} \to \mathcal{B}$, if $ax^{\mathcal{A}}\varphi = a\varphi x^{\mathcal{B}}$ for all $a \in A$ and $x \in X$. A surjective morphism is an *epimorphism*, and if there exists an epimorphism $\varphi : \mathcal{A} \to \mathcal{B}$, then $\mathcal{B}$ is an *image* of $\mathcal{A}$.

It is clear that if $\varphi : \mathcal{A} \to \mathcal{B}$ is a morphism of n.d. automata, then $Cx^{\mathcal{A}}\varphi =$

$C\varphi x^{\mathcal{B}}$ whenever $C \subseteq A$ and $x \in X$, and hence $aw^{\mathcal{A}}\varphi = a\varphi w^{\mathcal{B}}$ for all $a \in A$ and $w \in X^*$. Using this observation one proves easily the following proposition.

Proposition 4.3. *If $\varphi : \mathcal{A} \to \mathcal{B}$ is an epimorphism of n.d. automata, then $\mathrm{D}_i(\mathcal{A}) \subseteq \mathrm{D}_i(\mathcal{B})$, and hence any image of a* Di*-directable n.d. automaton is* Di*-directable ($i = 1, 2, 3$).*

The *direct product* of $\mathcal{A}$ and $\mathcal{B}$ is the n.d. automaton $\mathcal{A} \times \mathcal{B} = (A \times B, X)$ defined so that $(a, b)x^{\mathcal{A}\times\mathcal{B}} = ax^{\mathcal{A}} \times bx^{\mathcal{B}}$ for all $(a, b) \in A \times B$ and $x \in X$. Of course, this definition could be formulated more generally to give the direct product of n ($n \geq 0$) n.d. automata. It is easy to show that

$$(a, b)w^{\mathcal{A}\times\mathcal{B}} = aw^{\mathcal{A}} \times bw^{\mathcal{B}}$$

for all $(a, b) \in A \times B$ and $w \in X^*$. It is also obvious that $\mathcal{A} \times \mathcal{B}$ is complete iff $\mathcal{A}$ and $\mathcal{B}$ both are complete.

For ordinary automata the catenation uv of a directing word u of $\mathcal{A}$ and a directing word v of $\mathcal{B}$ is a directing word of $\mathcal{A} \times \mathcal{B}$. In the case of D1- and D3-directability this construction does not always work since $bu^{\mathcal{B}}$ may be empty for some $b \in B$, and it may fail even for complete D1-directable n.d. automata because $(au^{\mathcal{A}})v^{\mathcal{A}}$ is not necessarily a singleton set. Indeed, it is easy to show by examples that the classes **Dir**(1), **CDir**(1), **Dir**(2), **Dir**(3) and **Dir**(2)∩**Dir**(3) are not closed under direct products. For the two remaining classes the following positive results can be noted.

Proposition 4.4. *The direct product of two* D2*-directable c.n.d. automata is* D2*-directable, and the direct product of any two* D3*-directable c.n.d. automata is* D3*-directable.*

Proof. Let $\mathcal{A} = (A, X)$ and $\mathcal{B} = (B, X)$ be complete n.d. automata.

If $\mathcal{A}$ and $\mathcal{B}$ are D2-directable, then there are words $u, v \in X^*$ and subsets $C \subseteq A$ and $D \subseteq B$ such that $au^{\mathcal{A}} = C$ and $bv^{\mathcal{B}} = D$ for all $a \in A$ and $b \in B$. Then for all $(a, b) \in A \times B$,

$$(a, b)uv^{\mathcal{A}\times\mathcal{B}} = (C \times bu^{\mathcal{B}})v^{\mathcal{A}\times\mathcal{B}} = Cv^{\mathcal{A}} \times (bu^{\mathcal{B}})v^{\mathcal{B}} = Cv^{\mathcal{A}} \times D,$$

and hence $uv \in \mathrm{D}_2(\mathcal{A} \times \mathcal{B})$. Here we naturally need the fact that $bu^{\mathcal{B}} \neq \emptyset$ for all $b \in B$.

Assume now that $u \in \mathrm{D}_3(\mathcal{A})$ and $v \in \mathrm{D}_3(\mathcal{B})$, and that $c \in au^{\mathcal{A}}$ and $d \in bv^{\mathcal{B}}$ for all $a \in A$ and $b \in B$. Then for all $(a, b) \in A \times B$,

$$(a, b)uv^{\mathcal{A}\times\mathcal{B}} = (au^{\mathcal{A}})v^{\mathcal{A}} \times (bu^{\mathcal{B}})v^{\mathcal{B}}$$

contains the state (c', d), where c' is any given state from $cv^{\mathcal{A}}$.

5 Minimum-length directing words

If $\mathcal{A} = (A, X) \in \mathbf{Dir}(i)$ for some i, $1 \leq i \leq 3$, let

$$\mathrm{d}_i(\mathcal{A}) = \min\{\lg(w) : w \in \mathrm{D}_i(\mathcal{A})\}.$$

For all $i = 1, 2, 3$ and $n \geq 1$, we set

$$\mathrm{d}_i(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{Dir}(i),\ |A| = n\},$$

and

$$\mathrm{cd}_i(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{CDir}(i),\ |A| = n\},$$

Moreover, we denote by $\mathrm{d}(n)$ the usual maximal length of the minimum-length directing words of a deterministic n-state directable automaton as defined in [3], [4] or [8], for example.

It is clear that $\mathrm{d}(n) \leq \mathrm{cd}_i(n) \leq \mathrm{d}_i(n)$, for all $n \geq 1$ and $i = 1, 2, 3$. In [1] Burkhard proved that $\mathrm{cd}_1(n) = 2^n - n - 1$ for $n \geq 2$. To obtain this result he constructs for each $n \geq 2$ an n-state c.n.d. automaton for which the shortest D1-directing words are of length $2^n - n - 1$. On the other hand, he observes that if $w = x_1 \dots x_m x_{m+1}$ is a minimum-length D1-directing word of a c.n.d. automaton $\mathcal{A} = (A, X)$, then Aw is a singleton set and the sequence $Ax_1, \dots, Ax_1 \dots x_m$ consists of pairwise different subsets of A with at least two elements. Hence, $\lg(w) \leq 2^n - n - 1$. Since this observation is valid also for general n.d. automata, the functions $\mathrm{cd}_1(n)$ and $\mathrm{d}_1(n)$ are as follows. Moreover, the bound is accurate also for $n = 1$ since the empty word is a D1-directing word of any 1-state n.d. automaton.

Proposition 5.1. (Burkhard 1976) *For any* $n \geq 1$, $\mathrm{cd}_1(n) = \mathrm{d}_1(n) = 2^n - n - 1$.

In [6] Goralčik *et al.* proved that the number of factors needed to form a constant relation as a product of some given relations on an n-element set may grow exponentially with n. This result gives exponential lower bounds for $d_1(n)$ and $d_3(n)$. Since we already have an exact expression for $d_1(n)$, we use the example of [6], in a slightly modified form, to obtain a lower estimate for $d_3(n)$. By changing the construction suitably we obtain such lower bounds also for $d_2(n)$ and $cd_2(n)$.

For any $n \geq 2$, let $\omega(n)$ denote the maximal order of any permutation on the set $[n] = \{1, \dots, n\}$. In [6] it was shown that $\omega(n) \geq \lfloor \sqrt[3]{n} \rfloor!$ when n is the sum of the first k primes for some k. From this it is easy to infer that $\omega(n) \geq \lfloor \sqrt[3]{n} - 1 \rfloor!$ for every $n \geq 2$.

Proposition 5.2. For any $n \geq 2$,

$$\text{(a)} \qquad \lfloor \sqrt[3]{n} - 1 \rfloor! < cd_2(n) \leq \sum_{k=2}^{n} \binom{2^n - 1}{k},$$

and

(b) $$\lfloor \sqrt[3]{n} - 1 \rfloor ! < d_2(n) \leq \sum_{k=2}^{n} \binom{2^n}{k}.$$

Proof. First we establish both of the lower bounds. Since they are obviously valid for all small values of n, we may suppose that the permutations on $[n-1]$ of maximal order $\omega(n)$ consist of at least two cycles. Let σ be such a permutation and $C_1, \ldots, C_r$ its cycles. Obviously, we may also assume that the lengths $m_1, \ldots, m_r$ of these cycles are relative primes. Let us now define an n-state c.n.d. automaton $\mathcal{A} = ([n], \{x, y, z\})$ as follows.

Firstly, let $x^{\mathcal{A}} = \{(1, \sigma(1)), \ldots, (n-1, \sigma(n-1)), (n, n)\}$. In each cycle C_i we fix arbitrarily an element a_i and set $b_i = \sigma(a_i)$. Now $y^{\mathcal{A}}$ is defined so that $ay^{\mathcal{A}} = \{b_i\}$ if $a \in C_i$ $(1 \leq i \leq r)$, and $ny^{\mathcal{A}} = \{n\}$. Finally, $z^{\mathcal{A}}$ is defined so that

$$az^{\mathcal{A}} = \begin{cases} \{n\} & \text{if } a \in \{a_1, \ldots, a_r, n\}, \\ C_i & \text{if } a \in C_i \setminus \{a_i\},\ (1 \leq i \leq r). \end{cases}$$

Clearly, $nw^{\mathcal{A}} = \{n\}$ for all words $w \in \{x, y, z\}^*$. On the other hand, $aw^{\mathcal{A}} = \{n\}$ also for all other states $a \in [n-1]$ only in case we may write $w = uzv$, where u is such a word that for all $1 \leq i \leq r$ and $a \in C_i$, $au^{\mathcal{A}} = a_i$. It should now be clear that $yx^{m_1 m_2 \ldots m_r - 1}z$ is the shortest D2-directing word of $\mathcal{A}$, and its length is $\omega(n) + 1 > \lfloor \sqrt[3]{n} - 1 \rfloor !$.

The upper bounds are obtained simply by estimating in each case the number of non-final states of the recognizer $\mathbf{B}_2$ defined in the proof of Propsition 3.4.

Remark. For values of n less than $1331 = (11^3)$ the lower bounds of Proposition 5.2 are well below the bound $(n-1)^2$ given by Černý's well-known automata.

Proposition 5.3. *For any* $n \geq 1$, $(n-1)^2 \leq \mathrm{cd}_3(n) \leq \frac{1}{2}n(n-1)(n-2) + 1$.

Proof. Since the D3-directing words of an ordinary automaton are exactly its directing words, the lower bound is given by Černý's [3] well-known examples of n-state automata $(n \geq 1)$ for which the the shortest directing words are of length $(n-1)^2$.

A word w $(\in X^*)$ is said to D3-*merge* two distinct states a, b of an n.d. automaton $\mathcal{A} = (A, X)$ if $aw \cap bw \neq \emptyset$. We have the following lemmas.

Lemma 5.4. *A c.n.d. automaton* $\mathcal{A} = (A, X)$ *is* D3-*directable if and only if there is a* D3-*merging word for every pair of distinct states* $a, b \in A$.

Proof. The condition is necessary since any D3-directing word D3-merges every pair of states of $\mathcal{A}$. Suppose now that for each pair $a, b \in A$, $a \neq b$, there is a D3-merging word $w_{a,b}$, and let $A = \{1, \ldots, n\}$. We define inductively a sequence $\nu_0, \nu_1, \ldots, \nu_{n-1}$ of words as follows. For each $i = 1, \ldots, n-1$, let

$$M(i) = 1\nu_{i-1} \cap 2\nu_{i-1} \cap \ldots \cap i\nu_{i-1}.$$

1. Let $\nu_0 = \varepsilon$. Then $M(1) \neq \emptyset$.

2. Suppose that for some i, $1 \leq i \leq n-2$, we have defined a word ν_{i-1} ($\in X^*$) such that $M(i) \neq \emptyset$. If $M(i) \cap (i+1)\nu_{i-1} \neq \emptyset$, then let $\nu_i = \nu_{i-1}$. Otherwise, choose any $a \in M(i)$ and any $b \in (i+1)\nu_{i-1}$ and set $\nu_i = \nu_{i-1}w_{a,b}$. In both cases $M(i+1) \neq \emptyset$, and hence $\nu_{n-1} \in D_3(\mathcal{A})$.

Lemma 5.5. *Let $\mathcal{A} = (A, X)$ be an n-state n.d. automaton. If a pair $a, b \in A$, $a \neq b$ of states has a* D3*-merging word, then it has a* D3*-merging word of length $\leq \binom{n}{2}$.*

Proof. If $w = x_1 \dots x_k$ is a D3-merging word for $a, b \in A$, then there are sequences of states $a_0, a_1, \dots, a_k$ and $b_0, b_1, \dots, b_k$ such that

(1) $a_0 = a$, $b_0 = b$,

(2) $a_i \in a_{i-1}x_i$ and $b_i \in b_{i-1}x_i$ for all $i = 1, \dots, k$, and

(3) $a_k = b_k$.

If w is of a minimal length, the pairs $\{a_0, b_0\}, \dots, \{a_{k-1}, b_{k-1}\}$ are all distinct and $k \leq \binom{n}{2}$.

We may now complete the proof of Proposition 5.3. If a nontrivial c.n.d. automaton $\mathcal{A} = (A, X)$ is D3-directable, there must exist a pair of states $a, b \in A$, $a \neq b$ such that $ax \cap bx \neq \emptyset$ for some $x \in X$. By appending to such an x $n-2$ D3-merging words of length $\leq \binom{n}{2}$ as in the proof of Lemma 5.4 we get a D3-directing word of length $\leq 1 + (n-2)\binom{n}{2}$. It is clear that the bound is valid also for $n = 1$.

Proposition 5.6. *For any $n \geq 1$,*

$$\lfloor \sqrt[3]{n} - 1 \rfloor ! < \mathrm{d}_3(n) \leq \sum_{k=2}^{n} \binom{2^n - 1}{k} - \sum_{k=2}^{n} \sum_{r=1}^{m(k)} (-1)^{r-1} \binom{n}{r} \binom{2^{n-r}}{k},$$

where $m(k) = \max\{i : k \leq 2^{n-i}\}$.

Proof. For the lower bound it suffices to modify the construction of the automaton $\mathcal{A}$ used in the proof of Proposition 5.2 so that $az^{\mathcal{A}} = \emptyset$ if $a \in C_i \setminus \{a_i\}$ $(1 \leq i \leq r)$. The upper bound is obtained by considering any n-state D3-directable automaton $\mathcal{A} = (A, X)$ and estimating the number of possible non-final states of the recognizer $\mathbf{B}_3$ (defined in the proof of Proposition 3.4) from which a final state can be reached. First of all, we may discard all states containing the empty set. On the other hand, any state consisting of just one non-empty set is final. These two observations yield the first sum expression. From this number we should subtract the number of final states consisting of at least two subsets of A. Consider any k, $2 \leq k \leq n$. By the Principle of Inclusion and Exclusion the number of states $\{C_1, \dots, C_k\}$ of $\mathbf{B}_3$ such that $C_1 \cap \dots \cap C_k \neq \emptyset$ is given by

$$\binom{n}{1}\binom{2^{n-1}}{k} - \binom{n}{2}\binom{2^{n-2}}{k} + \dots + (-1)^{m(k)-1}\binom{n}{m(k)}\binom{2^{n-m(k)}}{k}.$$

The double sum to be subtracted from the first sum is now obtained by forming the sum of theses sums for $k = 2, \dots, n$.

References

[1] H.V. Burkhard, Zum Längenproblem homogener Experimente an deterministierten und nicht-deterministischen Automaten, *Elektronische Informationsverarbeitung und Kybernetik, EIK* **12** (1976), 301-306.

[2] A. Carpi, On synchronizing unambiguous automata, *Theoretical Computer Science* **60** (1988), 285-296.

[3] J. Černý, Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fysikalny Časopis SAV* **14** (1964), 208-215.

[4] J. Černý, A. Piricka & B. Rosenauerová, On directable automata, *Kybernetika (Praha)* **7** (1971), 289-297.

[5] F. Gécseg , B. Imreh, On Completeness of Nondeterministic Automata, *Acta Mathematica Hungarica* **68** (1995), 151-159.

[6] P. Goralčik, Z. Hedrlin, V. Koubek, J. Ryšlinková, A game of composing binary relations, *R.A.I.O. Informatique théorique/ Theoretical Informatics* **16** (1982), 365-369.

[7] G. Grätzer, *Universal Algebra*, 2nd edn. Springer-Verlag (New York Berlin Heidelberg Tokyo, 1979)

[8] B. Imreh, M. Steinby, Some Remarks on Directable Automata, *Acta Cybernetica* **12** (1995), 23-35.

[9] J.-E. Pin, Sur les mots synchronisants dans un automate fini, *Elekronische Informationsverarbeitung und Kybernetik, EIK* **14** (1978), 297-303.

[10] I. Rystsov, Reset words for commutative and solvable automata, *Theoretical Computer Science* **172** (1997), 273-279.

[11] P.H. Starke, *Abstrakte Automaten*, VEB Deutscher Verlag der Wissenschaften, Berlin 1969.

Acta Cybernetica 14 (1999) 117-133.

Syntactic Monoids of Codes*

H. Jürgensen†

Abstract

A general characterization theorem for syntactic monoids of codes that satisfy independence conditions of a special form is proved. This result provides insight in some known characterizations of classes of codes via syntactic monoids and provides a general mechanism for deriving new characterizations for other classes of languages.

1 Introduction

Many of the combinatorial properties of codes related to issues in information transmission, like synchronization delays or error resistance, can be expressed algebraically in terms of properties of the syntactic monoids of the codes themselves or of the syntactic monoids of the set of messages generated by the codes. Obtaining these algebraic characterizations is an important, but difficult problem.

Quite a few partial results have been obtained for various classes of codes, for example, for infix codes [15], outfix codes [9], and hypercodes [28], [30]. For a general overview, the reader should consult the books [2] and [25], the survey paper [12], and the other references listed at the end of this paper. A new characterization for infix codes and hypercodes has been obtained in [23]. In the present paper we extract and formulate a general characterization method from those specialized results which applies to all classes of codes that can be defined in a certain way. Thus, results analogous to those of [23] can now be obtained using this method for a large variety of classes of codes simply by proving that their definitions satisfy certain formal criteria.

While this method is applicable to any class of codes satisfying these criteria, its most elegant consequences seem to arise when it is applied to subclasses of the class of infix codes. We discuss the special cases of infix codes, infix codes which are also outfix codes, infix-shuffle codes of index n, hypercodes, solid codes, and reflective

*This research has been supported by the Natural Sciences and Engineering Research Council of Canada, Grant OGP0000243. An earlier version of this paper was presented at the Second International Colloquium on Words, Languages, and Combinatorics, Kyoto, 25–28 August, 1992 [11]. A summary of the results of the present paper is presented in Chapter 8 of [12].

†Dartment of Computer Science, The University of Western Ontario London, Ontario, Canada N6A 5B7 and Institut für Informatik Universität Potsdam Am Neuen Palais 10 D-14469 Potsdam, Germany, e-mail: helmut@uwo.ca.

codes. We also show why there is no such characterization for the intercodes of index 1, that is, the comma-free codes.

The paper is organized as follows: In Section 2, we introduce basic notions and notation. In Section 3, we investigate how conditions defining a class of codes are reflected in the syntactic monoids. In Sections 4 and 5, we then focus on classes of codes contained in the class of infix codes. Finally, Section 6 contains a few concluding remarks.

2 Basic Notions and Notation

In this section, we review the basic notions and introduce some notation. For further information regarding codes and their syntactic monoids the reader is referred to the books [2] and [25] and the survey paper [12].

An *alphabet* is a non-empty set.[1] Let X be an alphabet. Then X^* denotes the free monoid generated by X, that is, the set of all *words* over X, including the *empty word* 1, with concatenation as the multiplication.

Let M be a monoid. With every subset L of M one associates its *principal congruence* P_L given by

$$u \equiv v(P_L) \iff \big(\forall x, y \in M\ (xuy \in L \longleftrightarrow xvy \in L)\big).$$

The factor monoid M/P_L is the *syntactic monoid* of L and is denoted by $\operatorname{syn} L$. For $u \in M$, let $[u]_L$ denote the P_L-class of u. The canonical morphism of M onto $\operatorname{syn} L$, that is, the morphism $\sigma_L : u \mapsto [u]_L$, is called the *syntactic morphism*. The set L is said to be *disjunctive* if P_L is the equality relation. The *residue* of L is the set $W_L = \{u \mid u \in M \wedge MuM \cap L = \emptyset\}$.

A *language* over X is a subset of X^*. A language over a finite alphabet is said to be *regular* if it is accepted by a deterministic finite automaton. The syntactic monoid of a regular language is isomorphic with the transition monoid of the reduced complete finite automaton accepting the language.

A language L is said to be a *code* if the submonoid of X^*, which L generates, is freely generated by L. For the study of codes, the case of $|X| = 1$ is trivial and it is, therefore, common to assume without special mention that $|X| > 1$.

Let M be a monoid with zero element, $|M| \geq 2$. We denote the identity and the zero elements by 1 and 0, respectively. The intersection of all non-zero ideals, if it is different from $\{0\}$, is called the *core* of M, denoted by $\operatorname{core}(M)$. The set $\operatorname{annihil}(M) = \{c \mid \forall x \in M \setminus \{1\}\ xc = cx = 0\}$ is the set of *annihilators* of M.

A *pointed monoid*[2] is a pair (M, L) where M is a monoid and L is a subset of M. Let (M, L) and (M', L') be pointed monoids. A *pointed-monoid morphism* of (M, L) into (M', L') is a semigroup morphism φ of M into M' such that $\varphi^{-1}(L') = L$. Such a pointed-monoid morphism φ is *surjective, injective, bijective* if it is so as a

[1] In the literature on formal languages, alphabets are usually assumed to be finite. In this paper, the finiteness condition would not make an important difference. It is, therefore, omitted as in [23].

[2] Called *p-monoid* in [24].

semigroup morphism of M into M'; it is *non-erasing* if $\varphi^{-1}(1_{M'}) = \{1_M\}$. Let $\mathbb{P}$ denote the category of pointed monoids.

If (M, L) is a pointed monoid then σ_L is a surjective pointed-monoid morphism onto $(\operatorname{syn} L, \sigma_L(L))$. Moreover, if φ is a surjective pointed-monoid morphism of (M, L) onto (M', L') then there is a unique surjective pointed-monoid morphism ψ of (M', L') onto $(\operatorname{syn} L, \sigma_L(L))$ such that, for all $u \in M$, $\sigma_L(u) = \psi(\varphi(u))$.

A predicate P on $\mathbb{P}$ is said to be *invariant* if, for any pointed monoid (M, L), P satisfies the following conditions:

- For any surjective pointed-monoid morphism φ of (M, L), P is true on $(\varphi(M), \varphi(L))$ if P is true on (M, L).
- For any surjective non-erasing pointed-monoid morphism φ of (M, L), P is true on (M, L) if it is true on $(\varphi(M), \varphi(L))$.

The results of this paper are based on the following observation concerning predicates on the category of pointed monoids.

Theorem 1 *Let P be an invariant predicate on $\mathbb{P}$ and let $\mathcal{L}_P$ be the class of languages L over X for which P is true on (X^*, L). The following statements are true:*

(1) If σ_L is non-erasing then $L \in \mathcal{L}_P$ if and only if P is true on the pointed monoid $(\operatorname{syn} L, \sigma_L(L))$.

(2) If P is decidable on finite pointed monoids, L is (constructively) regular, and σ_L is non-erasing, then it is decidable whether $L \in \mathcal{L}_P$.

Proof: The first claim is just a restatement of the definition of invariance, applied to σ_L. For the second claim, if L is constructively given as a regular language then one can compute the syntactic monoid $\operatorname{syn} L$ and the set $\sigma_L(L)$. Note that σ_L is a pointed-monoid morphism. The fact that L is regular implies that $\operatorname{syn} L$ is finite. Therefore, P is decidable on $(\operatorname{syn} L, \sigma_L(L))$. The invariance of P implies that it is decidable whether $L \in \mathcal{L}_P$. □

To apply Theorem 1 to a given predicate P, one has to establish that P is invariant and decidable on finite pointed monoids. In this paper we focus on predicates on $\mathbb{P}$ which can be expressed in a special form called implicational independence condition.

3 Codes and Independence Conditions

Let X be an alphabet with $|X| > 1$. Many natural classes of codes over X are characterized by *independence conditions* on the free monoid X^*. A systematic study of this characterization method is presented in [17] and [18]; see also [12].

The independence conditions can be presented in various forms. The most general approach uses abstract dependence systems in the sense of universal algebra[3] with possible restrictions to finitely based dependence systems [12]. Incomparability with respect to certain partial orders, like the *prefix order*

$$u \underset{\mathrm{p}}{\leq} v \iff v \in uX^*,$$

as studied in [4], [14], and [25] is a quite special case of this approach. In this paper we only consider independence conditions that can be expressed in the form of implications involving equations over X^*.

Recall, for example, that the prefix order $\leq_{\mathrm{p}}$ defines the class $\mathcal{L}_{\mathrm{p}}$ of all prefix codes over X by

$$L \in \mathcal{L}_{\mathrm{p}} \iff L \subseteq X^+ \wedge \forall u, v \in L \left(u \underset{\mathrm{p}}{\leq} v \to u = v \right).$$

This condition could also be expressed as

$$L \subseteq X^+ \wedge \forall u, x \in X^* \ ((u \in L \wedge ux \in L) \to x = 1).$$

We now turn to defining this latter form of independence condition in more abstract terms.

Let M be an arbitrary monoid, let $\mathcal{M}$ be a finite set of subsets[4] of M, and let V be a set of variable symbols, such that $M \cap V = \emptyset$. Moreover, let Λ denote a set variable ranging over all the subsets[5] of M.

In the sequel we need to consider words built from elements of M and V, that is, words in $(M \cup V)^*$. An *equation* over M takes the form $u = v$ and an *inclusion* takes the form $u \in \Lambda$, where u and v are words over $(M \cup V)$. A *basic implicational independence condition* has the form

$$\langle \texttt{quantifier prefix} \rangle \big(\langle \texttt{formula} \rangle \longrightarrow \langle \texttt{formula} \rangle \big)$$

where the quantifier prefix and the formulæ satisfy the following conditions:

(I1) The quantifier prefix specifies, for each variable symbol in V, its range explicitly, that is, as one of the sets in $\mathcal{M}$. Moreover, the quantifier prefix may include quantification over the number of variables used. It involves only universal quantifiers.

(I2) The formulæ are disjunctions of conjunctions of equations and inclusions over M.

The first formula will be referred to as the *premiss,* the second one as the *conclusion* of the basic implicational independence condition.

[3] See [3], [5], and [6].

[4] In all our examples below we have $\mathcal{M} = \{M\}$.

[5] More precisely, the range of Λ is the set of all subsets of whichever monoid is being considered.

These still rather informal definitions can obviously be expressed rigorously.[6] An *implicational independence condition* is a conjunction of basic implicational independence conditions.

If I is an implicational independence condition on the monoid M and $L \subseteq M$, then I *is satisfied on* the pointed monoid (M, L) if I is true on M for $\Lambda = L$.

We list a few natural examples of implicational independence conditions for the case of $M = X^*$ where X is an alphabet with at least two elements and $\mathcal{M} = \{X^*\}$.

Example 1 *Let X be an alphabet with $|X| > 1$. In Table 1 we exhibit a list of implicational independence conditions defining classes of languages over X studied in the context of codes. Each implicational independence condition is identified by its property name which is also used to identify the corresponding class of languages. The prefix-shuffle codes, suffix-shuffle codes, infix-shuffle codes, and outfix-shuffle codes of index $n = 1$ are the prefix codes $\mathcal{L}_{\mathrm{p}}$, suffix codes $\mathcal{L}_{\mathrm{s}}$, infix codes $\mathcal{L}_{\mathrm{i}}$, and outfix codes $\mathcal{L}_{\mathrm{o}}$, respectively, in the usual sense.[7] The infix-shuffle codes of index n are called n-shuffle codes in [17] and elsewhere; in [19], [20] they are called n-infix codes; see also [22], [21]. The shuffle codes of index n as well as some of the other types of codes listed in Table 1 may look like mathematical artifacts; they do, however, capture important aspects of error detection; for details, see [12]. The relation between the classes of codes listed in Table 1 is shown in Figures 1 and 2. The n-codes shown there, for $n \geq 2$, are languages such that every subset of up to n elements is a code [7], [10]; the n-ps-codes are languages, such that every subset of up to n elements is a prefix code or a suffix code [8]. For details on infix and outfix codes and the classes $\mathcal{L}_{\mathrm{pi}}$ and $\mathcal{L}_{\mathrm{si}}$ see [9].*

Many natural classes of codes or code-related languages can be defined using implicational independence conditions. There are, however, some natural classes of codes characterized by finitely based dependence systems for which it seems to be impossible to express the independence condition in terms of an implicational independence condition; the class of uniform codes or block codes, that is, of codes all elements of which have the same length, seems to be an example of this kind.

Suppose that φ is a morphism of M onto a monoid M', and that I is an implicational independence condition on M. Then φ induces an implicational independence condition on M', denoted by $\varphi(I)$, as follows:

(I3) If I is a conjunction of basic implicational independence conditions then let $\varphi(I)$ be the conjunction of the images, under φ, of these basic implicational independence conditions. Let $\mathcal{M}'$ be the set of the images of the sets in $\mathcal{M}$. In the quantifier prefixes of I, replace any range in $\mathcal{M}$ by its image in $\mathcal{M}'$.

(I4) For a word u over $(M \cup V)$, let $\varphi(u)$ be the word over $(M' \cup V)$ obtained by mapping the elements of M into M' according to φ and leaving all variables

[6] In particular, quantification over the number of variables would need to be expanded.

[7] In some cases, as in that of the prefix codes, one would have to require explicitly that $L \subseteq X^+$ to rule out the trivial case of $L = \{1\}$ in which L is not a code. On the other hand, including this degenerate case allows for a simpler statement of the results.

Family	Name	Implicational condition
$\mathcal{L}_{\text{code}}$, codes	I_{code}	$\forall m \forall n \forall x_1, \ldots, x_m, y_1, \ldots, y_n \in X^*$ $((x_1 \in \Lambda \wedge \ldots \wedge x_m \in \Lambda \wedge y_1 \in \Lambda \wedge \ldots$ $\wedge\, y_n \in \Lambda \wedge x_1 \cdots x_m = y_1 \cdots y_n)$ $\rightarrow x_1 = y_1, \ldots, x_n = y_n)$
$\mathcal{L}_{\text{2-code}}$, 2-codes	$I_{\text{2-code}}$	$\forall u, v((u \in \Lambda \wedge v \in \Lambda \wedge uv = vu)$ $\rightarrow u = v)$
$\mathcal{L}_{\text{2-ps}}$, 2-ps-codes	$I_{\text{2-ps}}$	$\forall u, x, y\ ((u \in \Lambda \wedge ux \in \Lambda \wedge yu \in \Lambda$ $\wedge\, ux = yu) \rightarrow x = y = 1)$
$\mathcal{L}_{\text{p}_n}$, prefix-shuffle codes of index n	I_{p_n}	$\forall x_1, \ldots, x_n, y_1, \ldots, y_n$ $((x_1 \cdots x_n \in \Lambda$ $\wedge\, x_1 y_1 x_2 y_2 \cdots x_n y_n \in \Lambda)$ $\rightarrow y_1 = \cdots = y_n = 1)$
$\mathcal{L}_{\text{i}_n}, \mathcal{L}_{\text{o}_n}, \mathcal{L}_{\text{s}_n}$, infix-, outfix-, suffix-shuffle codes of index n	$I_{\text{i}_n}, I_{\text{o}_n}, I_{\text{s}_n}$	analogous to $\mathcal{L}_{\text{p}_n}$
$\mathcal{L}_{\text{p}_n} \cap \mathcal{L}_{\text{s}_n}$	$I_{\text{p}_n,\text{s}_n}$	$I_{\text{p}_n} \wedge I_{\text{s}_n}$
$\mathcal{L}_{\text{b}}$, bifix codes	I_{b}	see $\mathcal{L}_{\text{p}_n} \cap \mathcal{L}_{\text{s}_n}$ for $n = 1$
$\mathcal{L}_{\text{i}_n} \cap \mathcal{L}_{\text{o}_n}$	$I_{\text{i}_n,\text{o}_n}$	$I_{\text{i}_n} \wedge I_{\text{o}_n}$
$\mathcal{L}_{\text{h}}$, hypercodes	I_{h}	$\forall n \forall x_0, \ldots, x_n, y_1, \ldots, y_n \in X^*$ $((x_0 \cdots x_n \in \Lambda \wedge x_0 y_1 x_1 y_2 \cdots y_n x_n \in \Lambda)$ $\rightarrow y_1 = \cdots = y_n = 1)$
$\mathcal{L}_{\text{refl}}$, reflective languages	I_{refl}	$\forall x, y \in X^*\ (xy \in \Lambda \rightarrow yx \in \Lambda)$
$\mathcal{L}_{\text{pi}}$, p-infix codes	I_{pi}	$\forall u, x, y((u \in \Lambda \wedge xuy \in \Lambda) \rightarrow y = 1)$
$\mathcal{L}_{\text{si}}$, s-infix codes	I_{si}	$\forall u, x, y((u \in \Lambda \wedge xuy \in \Lambda) \rightarrow x = 1)$
$\mathcal{L}_{\text{inter}_n}$, intercodes of index n	I_{inter_n}	$\forall u_1, \ldots, u_{n+1}, v_1, \ldots, v_n, x, y$ $((u_1 \in \Lambda \wedge \cdots \wedge u_{n+1} \in \Lambda$ $\wedge\, v_1 \in \Lambda \wedge \cdots \wedge v_n \in \Lambda$ $\wedge\, u_1 \cdots u_{n+1} = x v_1 \cdots v_n y)$ $\rightarrow ((x = 1 \wedge y = u_{n+1})$ $\vee\, (x = u_1 \wedge y = 1)))$
$\mathcal{L}_{\text{ol-free}}$, overlap-free languages	$I_{\text{ol-free}}$	$\forall x, y, z\ ((xy \in \Lambda \wedge yz \in \Lambda)$ $\rightarrow (x = 1 \vee z = 1 \vee y = 1))$
$\mathcal{L}_{\text{solid}}$, solid codes	I_{solid}	$I_{\text{i}} \wedge I_{\text{ol-free}}$

Table 1: Implicational conditions for some of the language classes.

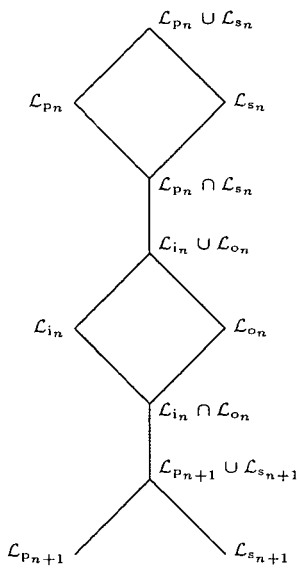


Figure 1: The relation between the shuffle codes.

unchanged.[8] For an equation $u = v$ over M let $\varphi(u = v)$ be the equation $\varphi(u) = \varphi(v)$ over M'. For an inclusion $u \in \Lambda$, let $\varphi(u \in \Lambda)$ be the inclusion $\varphi(u) \in \Lambda$. A formula over M is mapped by φ onto the corresponding formula of the images of the equations and inclusions.

This mechanism of induced implicational independence conditions is to be used in subsequent sections of this paper, to carry implicational independence conditions on X^* defining certain classes of languages into the syntactic monoids of these languages. Hence we consider the following two properties of an implicational independence condition I on a pointed monoid (M, L).

(I5) For any surjective pointed-monoid morphism φ of (M, L), if a premiss of I is false for some assignment of values to the variables then the image of that premiss is also false in $(\varphi(M), \varphi(L))$ for the corresponding value assignment.

(I6) For any surjective non-erasing pointed-monoid morphism φ of (M, L), if a conclusion of I is false for some assignment of values to the variables then the image of that conclusion is also false in $(\varphi(M), \varphi(L))$ for the corresponding value assignment.

[8] We assume that $M' \cap V = \emptyset$.

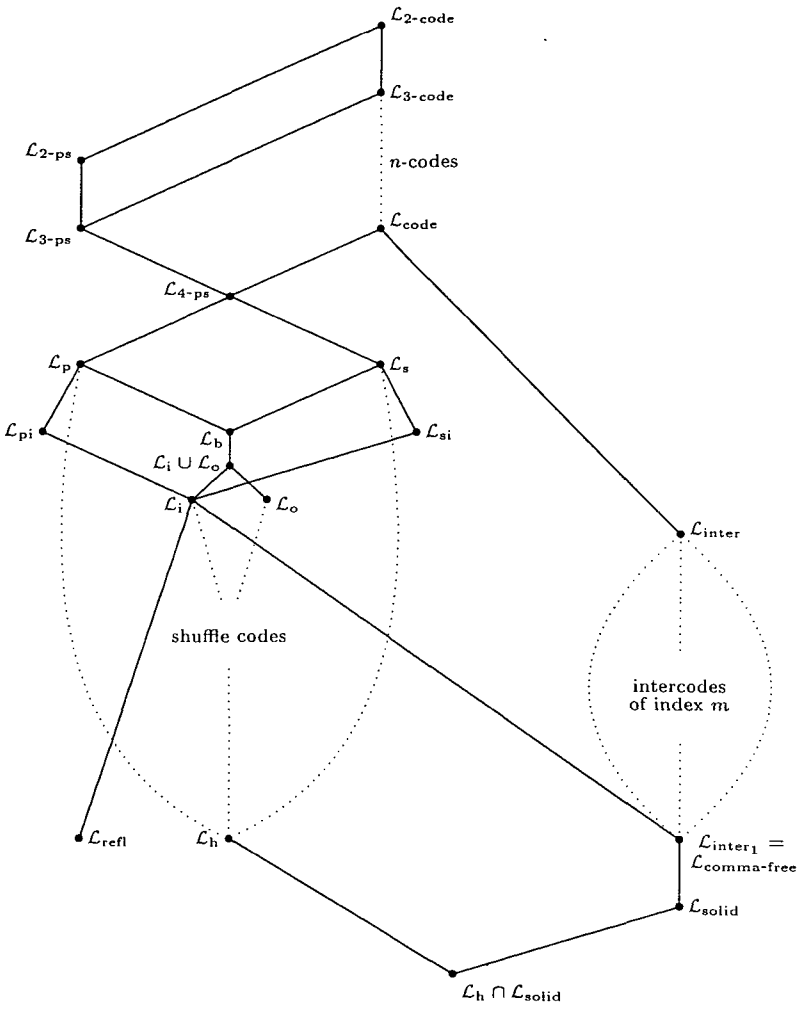


Figure 2: The relation between the classes of languages introduced in Table 1. Lines indicate (known) proper inclusions. Dotted lines indicate hierarchies. Intersections and unions are not, in general, indicated.

Theorem 2 *Let (M, L) be a pointed monoid and let I be an implicational independence condition on (M, L). Let φ be a surjective pointed-monoid morphism of (M, L). If I has property I5 and is satisfied on (M, L) then $\varphi(I)$ is satisfied on $(\varphi(M), \varphi(L))$. Conversely, when φ is also non-erasing, if I has property I6 and $\varphi(I)$ is satisfied on $(\varphi(M), \varphi(L))$ then I is satisfied on (M, L).*

Proof: Let (M, L) be a pointed monoid and φ be a surjective pointed-monoid morphism of (M, L) onto the pointed monoid (M', L'). As I is a conjunction of basic implicational independence conditions, I is satisfied on (M, L) if and only if each of its basic components is satisfied on (M, L). Therefore, we only need to prove the claim for I a basic implicational independence condition.

First note that if f is a formula appearing in I and f is true for some assignment of values to the variable symbols then $\varphi(f)$ is also true for the corresponding assignment of values. Indeed, if u and v are words over $(M \cup V)$ and $u = v$ for some value assignment then $\varphi(u) = \varphi(v)$ for the corresponding value assignment; this follows from the fact that φ is a semigroup morphism. Similarly, if $u \in L$ then $\varphi(u) \in \varphi(L) = L'$ as φ is surjective and $\varphi^{-1}(L') = L$, hence $\varphi(L) = L'$. Thus, as f is a disjunction of conjunctions of equations and inclusions, if f is true on (M, L) for some value assignment then also $\varphi(f)$ is true on (M', L') for the corresponding value assignment. Moreover, as φ is surjective, every value assignment in $(M'L')$ corresponds to – that is, is the image of – a value assignment in (M, L).

Let p and c be the premiss and conclusion of I, respectively, and suppose that I has property I5 and is satisfied on (M, L). Consider an assignment of values in M to the variable symbols occurring in p and c. If the premiss p is false under this assignment then also $\varphi(p)$ is false under the corresponding assignment in (M', L') by I5. On the other hand, if the premiss p is true under this assignment, then also the conclusion c must be true under this assignment. But then also $\varphi(p)$ and $\varphi(c)$ are true under the corresponding assignment. Hence, as the implication $p \to c$ is true under any value assignment also the implication $\varphi(p) \to \varphi(c)$ is true under any value assignment. Thus $\varphi(I)$ is satisfied on (M', L').

Conversely, assume that φ is also non-erasing and that I has property I6, and suppose $\varphi(I)$ is satisfied on (M', L'). Hence, for a value assignment either $\varphi(p)$ is false or both $\varphi(p)$ and $\varphi(c)$ are true. If $\varphi(p)$ is false then, for any value assignment in (M, L) that is mapped onto the given one by φ, also p is false. Suppose now that both $\varphi(p)$ and $\varphi(c)$ are true. Consider a value assignment in (M, L) which is mapped onto the given one by φ. If p is false under this assignment then $p \to c$ is true as needed. If p is true under this assignment then, by I6, c cannot be false as $\varphi(c)$ is true. This shows that I is satisfied on (M, L). □

An implicational independence condition I on a monoid M is said to be *free* if the only element of M occurring in I is 1 and if $\mathcal{M} = \{M\}$. A free implicational independence condition can be interpreted over any pointed monoid (M', L'). One treats M as a variable symbol for monoids, M having the value M' in this case, and 1 as denoting the identity element of the monoid under consideration. For a free implicational independence condition I let $\mathcal{L}_I$ be the class if pointed monoids (X^*, L), with X an alphabet, on which I is satisfied.

Theorem 3 *Let P be a predicate on $\mathbb{P}$, defined by a free implicational independence condition I. If I satisfies I5 and I6 for all pointed monoids then P is invariant.*

Proof: Consider a pointed monoid (M, L) and a surjective pointed-monoid morphism φ of (M, L) onto (M', L'). Suppose P is true on (M, L). Then I is satisfied on (M, L) as P is defined by I and I is free. By I5, using Theorem 2, $\varphi(I)$ is true on (M', L'). Thus P is true on (M', L').

For the converse, assume that φ is also non-erasing and that P is true on (M', L'). Then $\varphi(I)$ is satisfied on (M', L') and, by I6 and Theorem 2, I is satisfied on (M, L), hence P is true on (M, L). □

When I is a free implicational independence condition, instead of saying that $\varphi(I)$ is satisfied on $\varphi(M, L)$, it is more convenient to say that I is satisfied on $\varphi(M, L)$. Since this is unambiguous, we make this simplification in the sequel.[9]

Lemma 1 *Let (M, L) be a pointed monoid and let I be an implicational independence condition on M. The following statements hold true:*

(1) Suppose that, if a premiss p of I is false for some value assignment, then also $\sigma_L(p)$ is false for the corresponding value assignment. Then I satisfies I5.

(2) Suppose that σ_L is non-erasing and that, if a conclusion c of I is false for some value assignment, then also $\sigma_L(c)$ is false for the corresponding value assignment. Then I satisfies I6.

Proof: To prove (1), consider a surjective pointed-monoid morphism φ of (M, L) onto (M', L'). Then there is a surjective pointed-monoid morphism ψ of (M', L') onto $(\operatorname{syn} L, \sigma_L(L))$ such that $\sigma_L(u) = \psi(\varphi(u))$ for all $u \in M$. Suppose a premiss p of I is false; hence $\sigma_L(p)$ is false by assumption; if, however, $\varphi(p)$ were true then also $\psi(\varphi(p))$ would have to be true, a contradiction. Hence, $\varphi(p)$ is false. Thus I satisfies I5.

The proof of (2) is analogous; one only notes that σ_L being non-erasing implies that ψ has to be non-erasing. □

By Lemmma 1, it is sufficient to check I5 and I6 for syntactic morphisms. Invariance in general can be established by proving it for syntactic morphisms.

Lemma 2 *Let I be a free implicational independence condition. The following statements hold true:*

(1) If the premisses of I contain only inclusions then I satisfies I5 on any pointed monoid.

(2) If the conclusions of I contain only inclusions or equations of the form $u = 1$ then I satisfies I6 on any pointed monoid.

[9] For a completely rigorous treatment, one should build the category of pointed monoids from the category of monoids in such a way that the identity element is treated as a nullary operation symbol. In this way, a free implicational independence condition would not refer to any specific pointed monoid any more.

Proof: Let (M, L) and (M', L') be pointed monoids and let φ be a surjective pointed-monoid morphism of (M, L) onto (M', L').

First consider an inclusion $u \in \Lambda$. If this inclusion is true on (M', L') for some value assignment then, because of $\varphi^{-1}(L') = L$, this inclusion must be true on (M, L) for any pre-image, under φ, of this value assignment.

Suppose now that a premiss p of I consists solely of inclusions. If p is false for some value assignment on (M, L) then, by the preceding argument, $\varphi(p)$ must be false on (M', L').

Finally assume that φ is also non-erasing. Then $u = 1$ can only be true in (M', L') if it is true in (M, L). Thus, if a conclusion c of I consists only of inclusions and equations of the form $u = 1$ then, if it is false on (M, L), it is also false on (M', L'). □

We now examine some of the implicational independence conditions of Example 1 to determine which of these satisfy I5 or I5 and I6.

Theorem 4 *Let X be an alphabet with $|X| > 1$. Consider the implicational independence conditions listed in Example 1. The following statements hold true.*

(a) I_{code} and I_{inter_1} do not satisfy I5.

(b) I_{p_n}, I_{s_n}, I_{i_n}, I_{o_n}, $I_{\mathrm{i}_n,\mathrm{o}_n}$, $I_{\mathrm{p}_n,\mathrm{s}_n}$, I_{h}, I_{solid}, I_{refl} satisfy I5 and I6.

Proof: By Lemma 1 it is sufficient to consider syntactic morphisms.

We first prove statement (a): Let L be an infix code with $|L| > 1$. Clearly, L satisfies I_{code}. By [15], $\sigma_L(L)$ is a single element c in $\operatorname{syn} L$, and c is an annihilator different from 0. Let $u, v \in L$, $u \neq v$. Then $u^2 \neq v^2$. However, $\sigma_L(u^2) = c^2 = 0 = \sigma_L(v^2)$. Therefore, I_{code} does not satisfy I5.

Now consider I_{inter_1}. Every intercode of index 1 is an infix code [27]. Let L be an intercode of index 1. Therefore, the implicational independence condition $\sigma_L(I_{\mathrm{inter}_1})$ reduces to

$$\forall x, y \ \left(c^2 = xcy \rightarrow x = 1 \vee y = 1\right)$$

where x and y range over the syntactic monoid of L. As c is an annihilator, $c^2 = xcy = 0$ for all choices of x and y except $x = y = 1$. On the other hand, this premiss need not be true in X^*. For instance, let $X = \{a, b\}$ and $L = \{ab\}$; by [16], L is a solid code, hence an intercode of index 1. Let $x = y = a$. The only choice for u, v, w is $u = v = w = ab$. Hence $uv = abab \neq aaba = xwy$, but $\sigma_L(uv) = 0 = \sigma_L(xwy)$.

Statement (b) follows by Lemma 2. □

Note that the argument used to prove statement (b) does not apply in the cases of $\mathcal{L}_{\text{2-code}}$, $\mathcal{L}_{\text{2-ps}}$, or $\mathcal{L}_{\mathrm{inter}_n}$ as, in all these cases, the implicational independence condition provided in Example 1 contains an equation in the premiss.

Theorem 5 *Let M be a monoid such that $M \setminus \{1\}$ is a subsemigroup of M. Let I be a free implicational independence condition satisfying I5 and I6 on any pointed*

monoid. Then M is isomorphic with the syntactic monoid of a language L over an alphabet X such that I is satisfied on (X^*, L) if and only if M has a disjunctive subset L' such that I is satisfied on (M, L').

Proof: Suppose $L \subseteq X^*$ is such that I is satisfied on (X^*, L) and that $M \simeq \operatorname{syn} L$. Then $\sigma_L(L)$ is disjunctive in $\operatorname{syn} L$ and I is satisfied on $(\operatorname{syn} L, \sigma_L(L))$ by I5.

Conversely, let L' be a disjunctive subset of M such that I is satisfied on (M, L'). Let $X \subseteq M \setminus \{1\}$ be a set of generators of M. The embedding of X in M induces a morphism φ of X^* onto M. Let $L = \varphi^{-1}(L')$. Then $\varphi = \sigma_L$, φ is non-erasing, and I is satisfied on (X^*, L) by I6. □

Thus, many classes of languages defined by independence conditions have a characterization by syntactic monoids in the following sense: M is the syntactic monoid of such a language if and only if M contains a disjunctive subset which satisfies the conditions that characterize the respective class of languages. In the next sections of this paper, we apply this property to subclasses of the class of infix codes.

We conclude the present section with an interesting consequence of Theorem 5 regarding the decidability of language properties.

Theorem 6 *Let I be a free implicational independence condition satisfying I5 and I6 on all pointed monoids. Let X be an alphabet and let L be a regular language over X. If L is given effectively and σ_L is non-erasing then it is decidable whether I is satisfied on (X^*, L).*

Proof: Let L be a regular language, given in some effective way. Construct the reduced complete deterministic finite automaton A accepting L. Then $\operatorname{syn} L$ is isomorphic with the transition monoid of A. Moreover, one can compute $\sigma_L(L)$. Since $\operatorname{syn} L$ is finite one can check whether I is satisfied on $(\operatorname{syn} L, \sigma_L(L))$. If so I is satisfied on (X^*, L); otherwise it is not. □

With Theorem 6, we have a general proof of the decidability of certain code properties which, so far, has only been obtained for special cases with a special proof for each case. Another quite different general technique for proving such decidability results is provided in [13].

4 Infix Codes

In this section, we consider classes of codes "low in the hierarchy," that is, classes of codes contained in the class $\mathcal{C}_{\mathrm{i}}$, the class of infix codes. The syntactic monoids of infix codes have some special properties which render it particularly easy to express implicational independence conditions in them.

The following characterization of monoids which are isomorphic with syntactic monoids of infix codes is given in [23]. Note that some of the conditions imply that the monoid be subdirectly irreducible (see also [15]). In stating this result, we refer to the following list of properties of a monoid M.

(M_1) $M \setminus \{1\}$ is a subsemigroup of M.

(M_2) M has a zero.

(M_3) M has a disjunctive element c distinct from 1 and 0 such that $c = xcy$ implies $x = y = 1$.

(M_4) M has a disjunctive zero.

(M_5) There is an element $c \in \text{annihil}(M)$ distinct from 0 such that $\text{core}(M) = \{c, 0\}$.

(M_6) There is an element c distinct from 0 such that $c \in \text{core}(M) \cap \text{annihil}(M)$.

Theorem 7 [23] *The following conditions on a monoid M are equivalent.*

(1) M is isomorphic with the syntactic monoid of an infix code.

(2) M has the properties M_1, M_2, and M_3.

(3) M has the properties M_1, M_4, and M_5.

(4) M has the properties M_1, M_4, and M_6.

If L is an infix code and $c \in \text{syn}\, L$ is the element of condition M_3, M_5, or M_6 then $c = \sigma_L(L)$. Hence, condition M_4 states in particular that c satisfies the implicational independence condition I_i. This observation allows the following generalization of Theorem 7. For an arbitrary implicational independence condition, let $I(c)$ be the implicational independence condition obtained by substituting the symbols '$= c$' for every occurence of the symbols '$\in \Lambda$'.

Theorem 8 *Let I be an implicational independence condition satisfying I5 and I6 on all pointed monoids. If $\mathcal{L}_I \subseteq \mathcal{L}_i$ then the following conditions on a monoid M are equivalent.*

(1) M is isomorphic with the syntactic monoid syn L *of some L with $(X^*, L) \in \mathcal{L}_I$.*

(2) M has the properties M_1, M_2, M_3, and $I(c)$.

(3) M has the properties M_1, M_4, M_5, and $I(c)$.

(4) M has the properties M_1, M_4, M_6, and $I(c)$.

Proof: If M is isomorphic with the syntactic monoid syn L of some language L with $(X^*, L) \in \mathcal{L}_I$ then L is an infix code, and M has the properties M_1, M_2, and M_3. As $c = \sigma_L(L)$, also $I(c)$ holds true.

For infix codes, statements (2), (3), and (4) are equivalent by Theorem 7. Moreover, the proof shows that in each of M_3, M_5, and M_6, the element c is actually the same element of M. Hence, these statements are also equivalent for the class $\mathcal{L}_I$.

By Theorem 7, statement (4) implies that M is isomorphic with the syntactic monoid of an infix code L. Moreover, from the proof it follows that $\sigma_L(L) = c$. As $\varphi(I)$, for $\Lambda = L$, is equivalent to $I(c)$, it follows that I is satisfied on (X^*, L). □

Thus, Theorem 8 provides for a general method of characterizing the syntactic monoids of those classes of codes which are contained in the class of infix codes and are given by an implicational independence condition satisfying I5 and I6. In particular, this includes all the cases listed in Theorem 4(b) except the prefix and the suffix codes.

5 Infix and Outfix Codes

In [23], Corollary 1, a characterization of the syntactic monoids of hypercodes is derived which in part forms a special case of Theorem 8. In this section we show that also the remaining parts of that result can be obtained as a special case from a quite general theorem. For the result of [23] on hypercodes, the following observation is crucial: For a hypercode L, every P_L-class different from W_L is a hypercode. A similar statement can be made about other classes of codes contained in the class of outfix codes. See Figure 1 for the hierarchy of shuffle codes. In essence, we consider the classes contained in $\mathcal{L}_{\mathrm{i}} \cap \mathcal{L}_{\mathrm{o}} = \mathcal{L}_{\mathrm{i}_1} \cap \mathcal{L}_{\mathrm{o}_1}$. This class is characterized by the free implicational independence condition

$$\begin{aligned} &\forall u, x, y \in X^* \left(((u \in \Lambda \wedge xuy \in \Lambda) \to x = y = 1) \right. \\ &\left. \wedge\, ((xy \in \Lambda \wedge xuy \in \Lambda) \to u = 1) \right) \end{aligned}$$

which satisfies I5 and I6 on all pointed monoids. Thus, also the syntactic monoids of codes in $\mathcal{L}_{\mathrm{i}} \cap \mathcal{L}_{\mathrm{o}}$ can be characterized using Theorem 8.

Theorem 9 *The following statements hold true.*

(a) If L is an outfix code then every P_L-class different from the residue of L is an outfix code.

(b) If L is an infix-shuffle code of index n with $n \geq 3$ then every P_L-class different from the residue of L is an infix-shuffle code of index $n-2$.

(c) If L is a hypercode then every P_L-class different from the residue of L is a hypercode.

Proof: The proof of (a) is given in [9].

Let L be an infix-shuffle code of index n with $n \geq 3$, and consider P_L-equivalent words u, v such that u is not in the residue of L. Hence, there are s and t such that $sut \in L$ and, therefore, also $svt \in L$. Suppose that

$$u = u_1 u_2 \cdots u_{n-2} \quad \text{and} \quad v = v_1 u_1 v_2 u_2 \cdots v_{n-2} u_{n-2} v_{n-1}.$$

Letting $s = v_0$, $t = u_{n-1}$ and $v_n = 1$, one obtains $sut = svt$ from the fact that L is an infix-shuffle code of index n. This implies $u = v$. Thus, the class of u is an infix-shuffle code of index $n-2$.

The statement concerning hypercodes is proved in [23]. It is, of course, also an immediate consequence of (b) as a language is a hypercode if and only if it is an infix-shuffle code of index n for every n. □

By Theorem 9(b) and Figure 1, if $L \in \mathcal{L}_{\mathrm{i}} \cap \mathcal{L}_{\mathrm{o}}$, $L \in \mathcal{L}_{\mathrm{p}_n}$, $L \in \mathcal{L}_{\mathrm{s}_n}$, $L \in \mathcal{L}_{\mathrm{i}_n}$, $L \in \mathcal{L}_{\mathrm{o}_n}$, or $L \in \mathcal{L}_{\mathrm{h}}$ for $n \geq 2$, then every P_L-class different from the residue is an outfix code.

In view of Theorem 9, the result of [23] on hypercodes can be generalized significantly.

Theorem 10 *Let I be an implicational independence condition satisfying I5 and I6 on all pointed monoids and such that $\mathcal{L}_I \subseteq \mathcal{L}_{\mathrm{i}}$.*

Let I' be an implicational independence condition satisfying I5 and I6 on all elements of $\mathcal{L}_I$ and such that I implies I' and I' implies I_{i}.

Suppose that, if $(X^*, L) \in \mathcal{L}_I$, for every P_L-class L' different from W_L, I' is satisfied on (X^*, L'). Then the following conditions on a monoid M are equivalent.

(1) M is isomorphic with the syntactic monoid of a code in $\mathcal{L}_I$.

(2) M has the properties M_1, M_2, M_3, *and $I(c)$ and every element $x \in M \setminus \{0\}$ satisfies $I'(x)$.*

(3) M has the properties M_1, M_2, M_3, *and $I(c)$.*

Proof: The assumption about the P_L-classes different from W_L implies that $I'(x)$ holds true for every $x \in M \setminus \{0\}$. Thus, (1) implies (2). Obviously, statement (3) follows from (2). The remaining implication is already stated in Theorem 8. □

The case of hypercodes [23] is a special case of this result as are the cases of infix-shuffle codes of index n and of those infix codes which are also outfix codes.

6 Concluding Remarks

The main result of this paper is a general method for characterizing the syntactic monoids of codes when the class of codes is defined in a special formal way. The main application is to classes of codes, low in the hierarchy, that is, below the classes of infix codes and outfix codes.

The properties of infix codes and outfix codes that lead to particularly simple characterizations are the following: The syntactic monoid of an infix code has a disjunctive element. Every syntactic class of an outfix code is an outfix code. The obvious next step seems to be to abstract these properties and extend the results to higher regions of the hierarchy in possibly some restricted form.

References

[1] *Abstracts, Second International Colloquium on Words, Languages, and Combinatorics, Kyoto, 25–28 August, 1992.* Kyoto, 1992.

[2] J. Berstel, D. Perrin: *Theory of Codes.* Academic Press, Orlando, 1985.

[3] P. M. Cohn: *Universal Algebra.* D. Reidel Publishing Co., Dordrecht, revised ed., 1981.

[4] P. H. Day, H. J. Shyr: Languages defined by some partial orders. *Soochow J. Math.* **9** (1983), 53–62.

[5] F. Gécseg, H. Jürgensen: Algebras with dimension. *Algebra Universalis* **30** (1993), 422–446.

[6] F. Gécseg, H. Jürgensen: Dependence in algebras. *Fund. Inform.* **25** (1996), 247–256.

[7] M. Ito, H. Jürgensen, H. J. Shyr, G. Thierrin: Anti-commutative languages and n-codes. *Discrete Appl. Math.* **24** (1989), 187–196.

[8] M. Ito, H. Jürgensen, H. J. Shyr, G. Thierrin: n-Prefix-suffix languages. *Internat. J. Comput. Math.* **30** (1989), 37–56.

[9] M. Ito, H. Jürgensen, H. J. Shyr, G. Thierrin: Outfix and infix codes and related classes of languages. *J. Comput. System Sci.* **43** (1991), 484–508.

[10] M. Ito, H. Jürgensen, H. J. Shyr, G. Thierrin: Languages whose n-element subsets are codes. *Theoret. Comput. Sci.* **96** (1992), 325–344.

[11] H. Jürgensen: Syntactic monoids of codes. In [1], 108–112.

[12] H. Jürgensen, S. Konstantinidis: Codes. In G. Rozenberg, A. Salomaa (editors): *Handbook of Formal Language Theory.* Springer-Verlag, Berlin, 1997, vol. 1, 511–607.

[13] H. Jürgensen, K. Salomaa, S. Yu: Transducers and the decidability of independence in free monoids. *Theoret. Comput. Sci.* **134** (1994), 107–117.

[14] H. Jürgensen, H. J. Shyr, G. Thierrin: Codes and compatible partial orders on free monoids. In S. Wolfenstein (editor): *Algebra and Order, Proceedings of the 1st International Symposium on Ordered Algebraic Structures, Luminy-Marseilles, 1984.* 323–333, Heldermann-Verlag, Berlin, 1986.

[15] H. Jürgensen, G. Thierrin: Infix codes. In M. Arató, I. Kátai, L. Varga (editors): *Topics in the Theoretical Bases and Applications of Computer Science, Proceedings of the 4th Hungarian Computer Science Conference, Győr, 1985.* 25–29, Akadémiai Kiadó, Budapest, 1986.

[16] H. Jürgensen, S. S. Yu: Solid codes. *J. Inform. Process. Cybernet., EIK* **26** (1990), 563–574.

[17] H. Jürgensen, S. S. Yu: Relations on free monoids, their independent sets, and codes. *Internat. J. Comput. Math.* **40** (1991), 17–46.

[18] H. Jürgensen, S. S. Yu: Dependence systems and hierarchies of families of languages. Manuscript, 1996. In preparation.

[19] D. Y. Long: k-Outfix codes. *Chinese Ann. Math. Ser. A* **10** (1989), 94–99, in Chinese.

[20] D. Y. Long: k-Prefix codes and k-infix codes. *Acta Math. Sinica* **33** (1990), 414–421, in Chinese.

[21] D. Y. Long: n-Infix-outfix codes. In [1], 50–51.

[22] D. Y. Long: k-Bifix codes. *Riv. Mat. Pura Appl.* **15** (1994), 33–55.

[23] M. Petrich, G. Thierrin: The syntactic monoid of an infix code. *Proc. Amer. Math. Soc.* **109** (1990), 865–873.

[24] J. Sakarovitch: Un cadre algébrique pour l'étude des monoïdes syntactiques. In *Séminaire P. Dubreil (Algèbre), 28e année.* 14. Paris, 1974/75.

[25] H. J. Shyr: *Free Monoids and Languages.* Hon Min Book Company, Taichung, second ed., 1991.

[26] H. J. Shyr, G. Thierrin: Codes and binary relations. In M. P. Malliavin (editor): *Séminaire d'algèbre Paul Dubreil, Paris 1975–1976, (29ème Année). Lecture Notes in Computer Science* **586**, 180–188, Springer-Verlag, Berlin, 1977.

[27] H. J. Shyr, S. S. Yu: Intercodes and some related properties. *Soochow J. Math.* **16** (1990), 95–107.

[28] G. Thierrin: The syntactic monoid of a hypercode. *Semigroup Forum* **6** (1973), 227–231.

[29] G. Thierrin, S. S. Yu: Shuffle relations and codes. *J. Inform. and Optim. Sci.* **12** (1991), 441–449.

[30] E. Valkema: Syntaktische Monoide und Hypercodes. *Semigroup Forum* **13** (1976/77), 119–126.

[31] S. S. Yu: A characterization of intercodes. *Internat. J. Comput. Math.* **36** (1990), 39–45.

Acta Cybernetica **14** (1999) 135-149.

Tree Transducers and Formal Tree Series*

Werner Kuich †

Gécseg Ferenc barátomnak 60. születésnapja alkalmából

Abstract

We introduce tree transducers over formal tree series as a generalization of a restricted type of root-to-frontier tree transducers and show that linear nondeleting recognizable tree transducers do preserve recognizability of tree series.

1 Introduction and preliminaries

In this paper we give a uniform treatment of tree transducers and tree automata in terms of tree series and matrices.

In Section 2 we define tree transducers that map tree series into tree series. These tree transducers are a generalization of a restricted type of root-to-frontier tree transducers as described in Gécseg, Steinby [4, 5].

In Section 3 we consider linear and nondeleting tree representations and show certain algebraic properties of these tree representations.

In the last section we consider linear nondeleting recognizable tree transducers. Intuitively, these are generalizations of linear nondeleting root-to-frontier tree transducers with infinitely many productions whose right sides form recognizable tree languages. The main result of Section 4 is that linear nondeleting recognizable tree transducers do preserve recognizability of tree series. Our main result is a generalization of the following theorem of Thatcher [10]: *Linear root-to-frontier tree transducers preserve recognizability* (see also Gécseg, Steinby [4], Corollary IV.6.6).

It is assumed that the reader is familiar with the basics of semiring theory (see Kuich, Salomaa [9] and Kuich [6], Section 2). Throughout the paper, $\langle A, +, \cdot, 0, 1\rangle$ denotes a *commutative continuous* semiring. This means:

(o) the multiplication $\cdot$ is commutative;

*Partially supported by the "Stiftung Aktion Österreich-Ungarn".

†Abteilung für Theoretische Informatik Institut für Algebra und Diskrete Mathematik Technische Universität Wien, Wiedner Hauptstraße 8–10, A–1040 Wien e-mail: kuich@tuwien.ac.at

(i) A is partially ordered by the relation $\sqsubseteq$: $a \sqsubseteq b$ iff there exists a c such that $a + c = b$,

(ii) $\langle A, +, \cdot, 0, 1\rangle$ is a complete semiring,

(iii) $\sum_{i\in I} a_i = \sup(\sum_{i\in E} a_i \mid E \subseteq I, E \text{ finite})$, $a_i \in A$, $i \in I$, for an arbitrary index set I, where sup denotes the least upper bound with respect to $\sqsubseteq$.

In the sequel, we denote $\langle A, +, \cdot, 0, 1\rangle$ briefly by A.

Furthermore, $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \ldots \cup \Sigma_k \cup \ldots$ will always denote a *ranked alphabet*, where Σ_k, $k \geq 0$, contains the symbols of rank k and X will denote an *alphabet of leaf symbols*. By $T_\Sigma(X)$ we denote the set of *trees* formed by $\Sigma \cup X$. This set $T_\Sigma(X)$ is the smallest set formed according to the following conventions:

(i) if $\omega \in \Sigma_0 \cup X$ then $\omega \in T_\Sigma(X)$,

(ii) if $\omega \in \Sigma_k$, $k \geq 1$, and $t_1, \ldots, t_k \in T_\Sigma(X)$ then $\omega(t_1, \ldots, t_k) \in T_\Sigma(X)$.

If $\Sigma_0 \neq \emptyset$ then X may be the empty set.

By $A\langle\langle T_\Sigma(X)\rangle\rangle$ we denote the set of *formal tree series* over $T_\Sigma(X)$, i. e., the set of mappings $s : T_\Sigma(X) \to A$ written in the form $\sum_{t\in T_\Sigma(X)} (s,t)t$, where the coefficient (s,t) is the value of s for the tree $t \in T_\Sigma(X)$. For a formal tree series $s \in A\langle\langle T_\Sigma(X)\rangle\rangle$, we define the *support* of s, $\mathrm{supp}(s) = \{t \in T_\Sigma(X) \mid (s,t) \neq 0\}$. By $A\langle T_\Sigma(X)\rangle$ we denote the set of tree series in $A\langle\langle T_\Sigma(X)\rangle\rangle$ that have finite support. A power series with finite support is termed *polynomial*. (For more definitions see Kuich [7].)

Formal tree series induce continuous mappings called *substitutions* as follows. Let Y denote a set of variables, where $Y \cap (\Sigma \cup X) = \emptyset$ ($\emptyset$ denotes the empty set), and consider a mapping $h : Y \to A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$. This mapping can be extended to a mapping $h : T_\Sigma(X \cup Y) \to A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$ by $h(x) = x$, $x \in X$, and

$$h(\omega(t_1, \ldots, t_k)) = \bar{\omega}(h(t_1), \ldots, h(t_k)) = \sum_{t'_1,\ldots,t'_k \in T_\Sigma(X\cup Y)} (h(t_1), t'_1) \ldots (h(t_k), t'_k)\omega(t'_1, \ldots, t'_k)\,,$$

for $\omega \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(X \cup Y)$, $k \geq 0$. One more extension of h yields a mapping $h : A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle \to A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$ by defining $h(s) = \sum_{t\in T_\Sigma(X\cup Y)} (s,t)h(t)$. This last extension of h is a complete semiring morphism from $A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$ into $A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$. It is a continuous mapping (see Corollary 2.15 of Kuich [7]).

Let now $s \in A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$. Then, by definition, the formal tree series s *induces a mapping* $s : (A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle)^Y \to A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$ as follows: given $h : Y \to A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$, the value of s with argument h is simply $h(s)$, where h is the extended mapping. If $Y = \{y_1, \ldots, y_n\}$ is finite, we use the following notation: $h : Y \to A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$, where $h(y_i) = s_i$, $1 \leq i \leq n$, is denoted by $(s_i,\ 1 \leq i \leq n)$ or $(s_1, \ldots, s_n)$ and the value of s with argument h is denoted by $s(s_i,\ 1 \leq i \leq n)$ or $s(s_1, \ldots, s_n)$. Intuitively, this is simply the *substitution* of the formal tree series $s_i \in A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$ into the variables y_i,

$1 \leq i \leq n$, of $s \in A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$. The mapping $s : (A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle)^Y \to A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$, i. e., the substitution of formal tree series into the variables of Y, is a continuous mapping (see Theorem 2.18 of Kuich [7]). Observe that $s(s_1, \ldots, s_n) = \sum_{t \in T_\Sigma(X \cup Y)} (s,t)t(s_1, \ldots, s_n)$.

In certain situations, formulae are easier readable if we use the notation $s[s_i/y_i,\ 1 \leq i \leq n]$ for the substitution of the formal tree series s_i into the variables y_i, $1 \leq i \leq n$, of s instead of the notation $s(s_i,\ 1 \leq i \leq n)$. So we will use sometimes this notation $s[s_i/y_i,\ 1 \leq i \leq n]$.

In the same way, $s \in A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle$ also induces a mapping $s : (A\langle\langle T_\Sigma(X)\rangle\rangle)^Y \to A\langle\langle T_\Sigma(X)\rangle\rangle$.

Our tree automata and tree transducers will be defined by transition matrices. Let $Y_k = \{y_1, \ldots, y_k\}$, $k \geq 1$, and Y be sets of variables. A *matrix* $M \in (A\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle)^{I' \times I^k}$, $k \geq 1$, I' and I arbitrary index sets, *induces a mapping*

$$M : (A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle)^{I \times 1} \times \ldots \times (A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle)^{I \times 1} \to (A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle)^{I' \times 1}$$

(there are k argument vectors), defined by the entries of the resulting vector as follows: For $P_1, \ldots, P_k \in (A\langle\langle T_\Sigma(X \cup Y)\rangle\rangle)^{I \times 1}$ we define, for all $i \in I'$,

$$M(P_1, \ldots, P_k)_i = \sum_{i_1, \ldots, i_k \in I} M_{i,(i_1, \ldots, i_k)}((P_1)_{i_1}, \ldots, (P_k)_{i_k}) = \sum_{i_1, \ldots, i_k \in I} \sum_{t \in T_\Sigma(X \cup Y_k)} (M_{i,(i_1, \ldots, i_k)}, t)t((P_1)_{i_1}, \ldots, (P_k)_{i_k}) .$$

Throughout the whole paper, I (resp. Q) will denote an arbitrary (resp. a finite) index set.

2 Tree transducers

In this section we introduce tree transducers based on formal tree series and matrices. We show that these tree transducers are a generalization of a restricted type of root-to-frontier tree transducers as described in Gécseg, Steinby [4, 5].

In the sequel, Σ and Σ' denote finite ranked alphabets, X and X' denote leaf alphabets and $Z = \{z_i \mid i \geq 1\}$ denotes an alphabet of variables. We denote $Z_k = \{z_i \mid 1 \leq i \leq k\}$, $k \geq 1$, and $Z_0 = \emptyset$.

A *tree representation* (with state set Q) is a mapping μ from $\Sigma \cup X$ into matrices with entries in $A\langle\langle T_{\Sigma'}(X' \cup Z)\rangle\rangle$ such that

$$\mu : \Sigma_k \to (A\langle\langle T_{\Sigma'}(X' \cup Z_k)\rangle\rangle)^{Q \times Q^k},\ k \geq 1,$$
$$\mu : \Sigma_0 \cup X \to (A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q \times 1}.$$

For $f \in \Sigma_k$, $k \geq 1$, $\mu(f)$ *induces a mapping*

$$\mu(f) : (A\langle\langle T_{\Sigma'}(X' \cup Z)\rangle\rangle)^{Q \times 1} \times \cdots \times (A\langle\langle T_{\Sigma'}(X' \cup Z)\rangle\rangle)^{Q \times 1} \to (A\langle\langle T_{\Sigma'}(X' \cup Z)\rangle\rangle)^{Q \times 1}$$

(there are k argument vectors), defined by the entries of the resulting vector as follows: For $P_1, \ldots, P_k \in (A\langle\langle T_{\Sigma'}(X' \cup Z)\rangle\rangle)^{Q \times 1}$ and $q \in Q$, the mapping is

given by

$$\mu(f)(P_1,\ldots,P_k)_q = \sum_{q_1,\ldots,q_k\in Q} \mu(f)_{q,(q_1,\ldots,q_k)}((P_1)_{q_1},\ldots,(P_k)_{q_k})\,.$$

Observe that for $P_1,\ldots,P_k \in (A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1}$, the vector $\mu(f)(P_1,\ldots,P_k)$ is again in $(A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1}$. This means that

$$\langle(A\langle\langle T_{\Sigma'}(X'\cup Z)\rangle\rangle)^{Q\times 1},(\mu(f)\mid f\in\Sigma)\rangle \text{ and } \langle(A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1},(\mu(f)\mid f\in\Sigma)\rangle$$

are Σ-algebras. Hence, the mapping $\mu : X \to (A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1}$ can be uniquely extended to a morphism

$$\mu : T_\Sigma(X) \to (A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1}\,.$$

This morphic extension is defined inductively as follows:

$$\mu(f(t_1,\ldots,t_k)) = \mu(f)(\mu(t_1),\ldots,\mu(t_k))$$

for $f\in\Sigma_k$, $t_1,\ldots,t_k\in T_\Sigma(X)$.

A tree representation μ is called *polynomial* iff $\mu(f)\in(A\langle T_{\Sigma'}(X'\cup Z_k)\rangle)^{Q\times Q^k}$ for $f\in\Sigma_k$, $k\geq 1$, and $\mu(f)\in(A\langle T_{\Sigma'}(X')\rangle)^{Q\times 1}$ for $f\in\Sigma_0\cup X$. Observe that, for $|Q|=1$ and $A=\mathbb{B}$, our polynomial tree representations are nothing else than tree homomorphisms (see Gécseg, Steinby [5], page 18).

For $s\in A\langle\langle T_\Sigma(X)\rangle\rangle$ we define $\mu(s)=\sum_{t\in T_\Sigma(X)}(s,t)\otimes\mu(t)$, where $\otimes$ denotes the Kronecker product (see Kuich, Salomaa [9], Section 4). We are now in the position to define the notion of a tree transducer.

A *tree transducer* (with input alphabet Σ, input leaf alphabet X, output alphabet Σ', output leaf alphabet X')

$$\mathfrak{T}=(Q,\mu,S)$$

is given by

(i) a nonempty finite set Q of *states*,

(ii) a *tree representation* μ with state set Q,

(iii) $S\in(A\langle\langle T_{\Sigma'}(X'\cup Z_1)\rangle\rangle)^{1\times Q}$, called the *initial state vector*.

The *mapping* $||\mathfrak{T}||: A\langle\langle T_\Sigma(X)\rangle\rangle \to A\langle\langle T_{\Sigma'}(X')\rangle\rangle$ *realized by a tree transducer* $\mathfrak{T}=(Q,\mu,S)$ is defined by

$$||\mathfrak{T}||(s) = S(\mu(s)) = S(\sum_{t\in T_\Sigma(X)}(s,t)\otimes\mu(t))\,.$$

A tree transducer $\mathfrak{T}=(Q,\mu,S)$ is called *polynomial* iff μ is a polynomial tree representation, and the entries of S are of the form $S_q=a_qz_1$, $a_q\in A$, $q\in Q$.

We now connect our notion of tree transducer with the root-to-frontier tree transducers. By Gécseg, Steinby [5], a *root-to-frontier tree transducer*

$$\mathfrak{A} = (\Sigma, X, Q, \Sigma', X', P, Q')$$

is a system, where

(1) $\Sigma, \Sigma', X, X', Q$ are specified in the same way as in the definition of our tree transducer;

(2) P is a finite set of *productions* of the following types:
(i) $qx \to t$, where $q \in Q$, $x \in X$, $t \in T_{\Sigma'}(X')$;
(ii) $qf(z_1, \ldots, z_k) \to t$, where $q \in Q$, $f \in \Sigma_k$, $k \geq 0$, $t \in T_{\Sigma'}(X' \cup QZ_k)$;

(3) $Q' \subseteq Q$ is the set of *initial states*.

A root-to-frontier tree transducer $\mathfrak{A}$ is called *nondeterministically simple* iff for each production of type (ii) $qf(z_1, \ldots, z_k) \to t$ there exists a set $C_{qf \to t} = \{q_{i_1} z_1, \ldots, q_{i_k} z_k\}$ such that $t \in T_{\Sigma'}(X' \cup C_{qf \to t})$. (Compare with Gécseg, Steinby) [4], Exercise 4 on page 213.) Observe that not all elements $q_{i_1} z_1, \ldots, q_{i_k} z_k$ of $C_{qf \to t}$ have to appear in t, i. e., $\mathfrak{A}$ needs not to be nondeleting.

For the forthcoming considerations in this section, our basic semiring is the Boolean semiring $\mathbb{B}$ and we use without mentioning the isomorphism between $\mathbb{B}\langle\langle T_\Sigma(X)\rangle\rangle$ and $\mathfrak{P}(T_\Sigma(X))$. Given a nondeterministically simple root-to-frontier tree transducer $\mathfrak{A} = (\Sigma, X, Q, \Sigma', X', P, Q')$, we construct a polynomial tree transducer $\mathfrak{T} = (Q, \mu, S)$ that behaves analogous to $\mathfrak{A}$. The polynomial tree representation μ is defined as follows:

(i) For $x \in X$, $q \in Q$, $t \in T_{\Sigma'}(X')$, $(\mu(x)_q, t) = 1$ if $qx \to t \in P$.

(ii) For $f \in \Sigma_k$, $q, q_{i_1}, \ldots, q_{i_k} \in Q$, $t(z_1, \ldots, z_k) \in T_{\Sigma'}(X' \cup Z_k)$, $k \geq 0$, $(\mu(f)_{q,(q_{i_1}, \ldots, q_{i_k})}, t(z_1, \ldots, z_k)) = 1$ if $qf(z_1, \ldots, z_k) \to t(q_{i_1} z_1, \ldots, q_{i_k} z_k) \in P$ and $C_{qf \to t} = \{q_{i_1} z_1, \ldots, q_{i_k} z_k\}$.

The initial state vector S is defined by $S_q = z_1$ if $q \in Q'$, $S_q = 0$ if $q \in Q - Q'$.

We claim that, for $s \in T_\Sigma(X)$, $t \in T_{\Sigma'}(X')$ and $q \in Q$

$$(\mu(s)_q, t) = 1 \text{ iff } qs \Rightarrow^* t$$

and prove it by induction on the form of trees in $T_\Sigma(X)$. Clearly, the claim holds true for trees in $X \cup \Sigma_0$. Let now $f \in \Sigma_k$ for some $k \geq 1$, $s_1, \ldots, s_k \in T_\Sigma(X)$, and $t = f(s_1, \ldots, s_k)$. By induction hypothesis, we have $(\mu(s_j)_{q_j}, t_j) = 1$ iff $q_j s_j \Rightarrow^* t_j$, $q_j \in Q$, $t_j \in T_{\Sigma'}(X')$, $1 \leq j \leq k$. Let now $(\mu(f(s_1, \ldots, s_k))_q, t) = 1$, i. e., for some $q_1, \ldots, q_k \in Q$

$$(\mu(f)_{q,(q_1, \ldots, q_k)}(\mu(s_1)_{q_1}, \ldots, \mu(s_k)_{q_k}), t) = 1\,.$$

Then there exist $v \in T_{\Sigma'}(X' \cup Z_k)$ and $t_1, \ldots, t_k \in T_{\Sigma'}(X')$ such that $(\mu(f)_{q,(q_1,\ldots,q_k)}, v) = 1$, $(\mu(s_j)_{q_j}, t_j) = 1$, $1 \leq j \leq k$, and $t = v(t_1, \ldots, t_k)$. This implies

$$qf(s_1, \ldots, s_k) \Rightarrow^* v(q_1 s_1, \ldots, q_k s_k) \Rightarrow^* v(t_1, \ldots, t_k) = t\,.$$

Similarly, we can show that $qf(s_1, \ldots, s_k) \Rightarrow^* t$ implies $(\mu(f(s_1, \ldots, s_k))_q, t) = 1$. This yields our first theorem.

Theorem 1 *Let $\mathfrak{A}$ be a nondeterministically simple root-to-frontier tree transducer and $\mathfrak{T}$ be the polynomial tree transducer constructed from $\mathfrak{A}$. Let $L \subseteq T_\Sigma(X)$ be a tree language. Then $\mathfrak{A}$ maps L to the tree language $supp(\|\mathfrak{T}\|(char(L))) \subseteq T_{\Sigma'}(X')$.*

3 Linear and nondeleting tree transducers

In this section we introduce linear and nondeleting tree representations and tree transducers. The main result of this section is that a linear nondeleting representation can be extended to a morphism over matrices of formal tree series.

A tree $t \in T_\Sigma(X \cup Z_k)$, $k \geq 1$, is called *linear* iff the variable z_j appears at most once in t, $1 \leq j \leq k$. A tree $t \in T_\Sigma(X \cup Z_k)$, $k \geq 1$, is called *nondeleting* iff the variable z_j appears at least once in t, $1 \leq j \leq k$. A tree series $s \in A\langle\langle T_\Sigma(X \cup Z_k)\rangle\rangle$, $k \geq 1$, is called *linear* or *nondeleting* iff all $t \in \mathrm{supp}(s)$ are linear or nondeleting, respectively. A tree representation μ is called *linear* or *nondeleting* iff all entries of $\mu(f)$, $f \in \Sigma_k$, $k \geq 1$, are linear or nondeleting tree series, respectively. A tree transducer $\mathfrak{T} = (Q, \mu, S)$ is called *linear* or *nondeleting* iff μ is linear or nondeleting, respectively, and the entries of S are of the form $S_q = a_q z_1$, $a_q \in A$, $q \in Q$.

Before we can state and prove our main result of this section we need a series of technical lemmas.

Lemma 2 *Let, for some $k \geq 1$, $t \in T_\Sigma(X \cup Z_k)$, and $s_j \in A\langle\langle T_\Sigma(X)\rangle\rangle$, $a_j \in A$, $1 \leq j \leq k$. Assume that the variable z_j appears $m_j \geq 0$ times in t, $1 \leq j \leq k$. Then*

$$t(a_1 s_1, \ldots, a_k s_k) = a_1^{m_1} \ldots a_k^{m_k} t(s_1, \ldots, s_k)\,.$$

Proof. The proof is by induction on the form of trees in $T_\Sigma(X \cup Z_k)$. The lemma is trivial for $t = x$ or $t = z_j$, $1 \leq j \leq k$. Let now t be of the form $f(t_1, \ldots, t_m)$, where $f \in \Sigma_m$ for some $m \geq 1$, and $t_1, \ldots, t_m \in T_\Sigma(X \cup Z_k)$. Let z_j appear u_{ij} times in t_i, $1 \leq j \leq k$, $1 \leq i \leq m$. Then we have by induction hypothesis $t_i(a_1 s_1, \ldots, a_k s_k) = a_1^{u_{i1}} \ldots a_k^{u_{ik}} t_i(s_1, \ldots, s_k)$, $1 \leq i \leq m$. Hence, $t(a_1 s_1, \ldots, a_k s_k) = f(t_1(a_1 s_1, \ldots, a_k s_k), \ldots, t_m(a_1 s_1, \ldots, a_k s_k)) = f(a_1^{u_{11}} \ldots a_k^{u_{1k}} t_1(s_1, \ldots, s_k), \ldots, a_1^{u_{m1}} \ldots a_k^{u_{mk}} t_m(s_1, \ldots, s_k)) = a_1^{u_{11}+\cdots+u_{m1}} \ldots a_k^{u_{1k}+\cdots+u_{mk}} f(t_1(s_1, \ldots, s_k), \ldots, t_m(s_1, \ldots, s_k)) = a_1^{u_{11}+\cdots+u_{m1}} \ldots a_k^{u_{1k}+\cdots+u_{mk}} t(s_1, \ldots, s_k)$. □

Lemma 3 *Let, for some $k \geq 1$, $s \in A\langle\langle T_\Sigma(X \cup Z_k)\rangle\rangle$ be linear and nondeleting, and $s_j \in A\langle\langle T_\Sigma(X)\rangle\rangle$, $a_j \in A$ for $1 \leq j \leq k$. Then*

$$s(a_1 s_1, \ldots, a_k s_k) = a_1 \ldots a_k s(s_1, \ldots, s_k) .$$

Proof. We have $s = \sum_{t \in T_\Sigma(X \cup Z_k)} (s,t)t$. Since s is linear and nondeleting, we obtain, by Lemma 2, $t(a_1 s_1, \ldots, a_k s_k) = a_1 \ldots a_k t(s_1, \ldots, s_k)$ for all $t \in T_\Sigma(X \cup Z_k)$ such that $(s,t) \neq 0$. Hence $s(a_1 s_1, \ldots, a_k s_k) = \sum_{t \in T_\Sigma(X \cup Z_k)} a_1 \ldots a_k (s,t)$ $t(s_1, \ldots, s_k) = a_1 \ldots a_k s(s_1, \ldots, s_k)$. □

Lemma 4 *Assume that the variable z_1 appears exactly once in $t \in T_\Sigma(X \cup Z_k)$, $k \geq 1$. Let $s_i \in A\langle\langle T_\Sigma(X)\rangle\rangle$ for $i \in I$ and $r_2, \ldots, r_k \in A\langle\langle T_\Sigma(X)\rangle\rangle$. Then*

$$t(\sum_{i \in I} s_i, r_2, \ldots, r_k) = \sum_{i \in I} t(s_i, r_2, \ldots, r_k) .$$

Proof. The proof is by induction on the form of trees in $T_\Sigma(X \cup Z_k)$. The lemma is trivial for $t = z_1$. Let now t be of the form $f(t_1, \ldots, t_m)$, where $f \in \Sigma_m$, and $t_1, \ldots, t_m \in T_\Sigma(X \cup Z_k)$. The variable z_1 appears in exactly one of the subtrees t_j, $1 \leq j \leq k$, say in t_u. By induction hypothesis we have $t_u(\sum_{i \in I} s_i, r_2, \ldots, r_k) = \sum_{i \in I} t_u(s_i, r_2, \ldots, r_k)$. Hence,

$$\begin{aligned} &f(t_1(\textstyle\sum_{i \in I} s_i, r_2, \ldots, r_k), \ldots, t_u(\sum_{i \in I} s_i, r_2, \ldots, r_k), \ldots, \\ &\qquad\qquad t_m(\textstyle\sum_{i \in I} s_i, r_2, \ldots, r_k)) = \\ &f(t_1(r_1, r_2, \ldots, r_k), \ldots, \textstyle\sum_{i \in I} t_u(s_i, r_2, \ldots, r_k), \ldots, t_m(r_1, r_2, \ldots, r_k)) = \\ &\textstyle\sum_{i \in I} f(t_1(r_1, r_2, \ldots, r_k), \ldots, t_u(s_i, r_2, \ldots, r_k), \ldots, t_m(r_1, r_2, \ldots, r_k)) \end{aligned}$$

for all $r_1 \in A\langle\langle T_\Sigma(X)\rangle\rangle$. Hence, the last sum is equal to

$$\begin{aligned} &\textstyle\sum_{i \in I} f(t_1(s_i, r_2, \ldots, r_k), \ldots, t_u(s_i, r_2, \ldots, r_k), \ldots, t_m(s_i, r_2, \ldots, r_k)) = \\ &\textstyle\sum_{i \in I} (f(t_1, \ldots, t_u, \ldots, t_m))(s_i, r_2, \ldots, r_k) = \sum_{i \in I} t(s_i, r_2, \ldots, r_k) . \end{aligned}$$

□

Lemma 5 *Let, for some $k \geq 1$, $s \in A\langle\langle T_\Sigma(X \cup Z_k)\rangle\rangle$ be linear and nondeleting, and $s_i \in A\langle\langle T_\Sigma(X)\rangle\rangle$, for $i \in I$. Moreover, let $r_2, \ldots, r_k \in A\langle\langle T_\Sigma(X)\rangle\rangle$. Then*

$$s(\sum_{i \in I} s_i, r_2, \ldots, r_k) = \sum_{i \in I} s(s_i, r_2, \ldots, r_k) .$$

Proof. We have, by Lemma 4, $s(\sum_{i \in I} s_i, r_2, \ldots, r_k) = \sum_{t \in T_\Sigma(X \cup Z_k)} (s,t) t(\sum_{i \in I} s_i, r_2, \ldots, r_k) =$ $\sum_{i \in I} \sum_{t \in T_\Sigma(X \cup Z_k)} (s,t) t(s_i, r_2, \ldots, r_k) = \sum_{i \in I} s(s_i, r_2, \ldots, r_k)$. □

Clearly, Lemma 5 also holds for argument places different from one.

Theorem 6 *Let, for some $k \geq 1$, $s \in A\langle\langle T_\Sigma(X \cup Z_k)\rangle\rangle$ be linear and nondeleting, and $s_{i_j} \in A\langle\langle T_\Sigma(X)\rangle\rangle$, $a_{i_j} \in A$ for $i_j \in I_j$, $1 \leq j \leq k$. Then*

$$s(\sum_{i_1 \in I_1} a_{i_1} s_{i_1}, \ldots, \sum_{i_k \in I_k} a_{i_k} s_{i_k}) = \sum_{i_1 \in I_1} \ldots \sum_{i_k \in I_k} a_{i_1} \ldots a_{i_k} s(s_{i_1}, \ldots, s_{i_k}) .$$

Proof. By Lemmas 3 and 5. □

Given a tree representation μ, we now extend μ to mappings

$$\mu : (A\langle\langle\Sigma_k\rangle\rangle)^{I\times I^k} \to ((A\langle\langle T_{\Sigma'}(X'\cup Z_k)\rangle\rangle)^{Q\times Q^k})^{I\times I^k} \text{ for } k\geq 0\,,$$

and

$$\mu : (A\langle\langle T_\Sigma(X)\rangle\rangle)^{I\times 1} \to ((A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1})^{I\times 1}$$

by

$$\mu(M) = \sum_{f\in\Sigma_k} (M,f)\otimes\mu(f), \quad M\in(A\langle\langle\Sigma_k\rangle\rangle)^{I\times I^k},\ k\geq 0\,,$$

and

$$\mu(P) = \sum_{t\in T_\Sigma(X)} (P,t)\otimes\mu(t), \quad P\in(A\langle\langle T_\Sigma(X)\rangle\rangle)^{I\times 1}\,.$$

Observe that $\mu(M)$, $M\in(A\langle\langle\Sigma_k\rangle\rangle)^{I\times I^k}$, $k\geq 0$, induces a mapping $\mu(M) : ((A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1})^{I\times 1}\times\ldots\times((A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1})^{I\times 1}\to((A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1})^{I\times 1}$ (there are k argument vectors). The next theorem implies that a linear nondeleting tree representation μ is a morphism from the Σ-algebra

$$\langle(A\langle\langle T_\Sigma(X)\rangle\rangle)^{I\times 1}, (M_f \mid f\in\Sigma)\rangle, \text{ for some } M_f\in(A\langle\langle\Sigma_k\rangle\rangle)^{I\times I^k},\ f\in\Sigma_k,\ k\geq 0\,,$$

into the Σ-algebra

$$\langle((A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q\times 1})^{I\times 1}, (\mu(M_f)\mid f\in\Sigma)\rangle\,.$$

Theorem 7 *Let $M\in(A\langle\langle\Sigma_k\rangle\rangle)^{I\times I^k}$, $P_1,\ldots,P_k\in(A\langle\langle T_\Sigma(X)\rangle\rangle)^{I\times 1}$ for some $k\geq 1$, and μ be a linear nondeleting tree representation with state set Q. Then*

$$\mu(M)(\mu(P_1),\ldots,\mu(P_k)) = \mu(M(P_1,\ldots,P_k))\,.$$

Proof. We first compute the left side of the equality of the theorem for indices $i\in I$ and $q\in Q$:

$$\begin{aligned}
&(\mu(M)(\mu(P_1),\ldots,\mu(P_k))_i)_q = \\
&\textstyle\sum_{i_1,\ldots,i_k\in I}\sum_{q_1,\ldots,q_k\in Q}(\mu(M)_{i,(i_1,\ldots,i_k)})_{q,(q_1,\ldots,q_k)} \\
&\qquad((\mu(P_1)_{i_1})_{q_1},\ldots,(\mu(P_k)_{i_k})_{q_k}) = \\
&\textstyle\sum_{i_1,\ldots,i_k\in I}\sum_{q_1,\ldots,q_k\in Q}((\sum_{f\in\Sigma_k}(M,f)\otimes\mu(f))_{i,(i_1,\ldots,i_k)})_{q,(q_1,\ldots,q_k)} \\
&(((\textstyle\sum_{t_1\in T_\Sigma(X)}(P_1,t_1)\otimes\mu(t_1))_{i_1})_{q_1},\ldots,((\sum_{t_k\in T_\Sigma(X)}(P_k,t_k)\otimes\mu(t_k))_{i_k})_{q_k}) = \\
&\textstyle\sum_{i_1,\ldots,i_k\in I}\sum_{q_1,\ldots,q_k\in Q}\sum_{f\in\Sigma_k}(M,f)_{i,(i_1,\ldots,i_k)}\mu(f)_{q,(q_1,\ldots,q_k)} \\
&\qquad(\textstyle\sum_{t_1\in T_\Sigma(X)}(P_1,t_1)_{i_1}\mu(t_1)_{q_1},\ldots,\sum_{t_k\in T_\Sigma(X)}(P_k,t_k)_{i_k}\mu(t_k)_{q_k}) = \\
&\textstyle\sum_{i_1,\ldots,i_k\in I}\sum_{q_1,\ldots,q_k\in Q}\sum_{f\in\Sigma_k}\sum_{t_1,\ldots,t_k\in T_\Sigma(X)}(M,f)_{i,(i_1,\ldots,i_k)} \\
&\qquad(P_1,t_1)_{i_1}\cdots(P_k,t_k)_{i_k}\mu(f)_{q,(q_1,\ldots,q_k)}(\mu(t_1)_{q_1},\ldots,\mu(t_k)_{q_k})\,.
\end{aligned}$$

Here the last equality follows by Theorem 6. We now compute the right side of the equality of the theorem for indices $i \in I$ and $q \in Q$:

$$
\begin{aligned}
&(\mu(M(P_1,\ldots,P_k))_i)_q = \\
&\textstyle\sum_{t\in T_\Sigma(X)}(M(P_1,\ldots,P_k),t)_i\mu(t)_q = \\
&\textstyle\sum_{t\in T_\Sigma(X)}(\sum_{i_1,\ldots,i_k\in I}(M_{i,(i_1,\ldots,i_k)}((P_1)_{i_1},\ldots,(P_k)_{i_k}),t))\mu(t)_q = \\
&\textstyle\sum_{t\in T_\Sigma(X)}\sum_{i_1,\ldots,i_k\in I}\sum_{f\in\Sigma_k}((M_{i,(i_1,\ldots,i_k)},f)f((P_1)_{i_1},\ldots,(P_k)_{i_k}),t)\mu(t)_q = \\
&\textstyle\sum_{t_1,\ldots,t_k\in T_\Sigma(X)}\sum_{i_1,\ldots,i_k\in I}\sum_{f\in\Sigma_k}(M_{i,(i_1,\ldots,i_k)},f) \\
&\qquad\qquad((P_1)_{i_1},t_1)\ldots((P_k)_{i_k},t_k)\mu(f(t_1,\ldots,t_k))_q = \\
&\textstyle\sum_{t_1,\ldots,t_k\in T_\Sigma(X)}\sum_{i_1,\ldots,i_k\in I}\sum_{f\in\Sigma_k}\sum_{q_1,\ldots,q_k\in Q}(M_{i,(i_1,\ldots,i_k)},f) \\
&\qquad\qquad((P_1)_{i_1},t_1)\ldots((P_k)_{i_k},t_k)\mu(f)_{q,(q_1,\ldots,q_k)}(\mu(t_1)_{q_1},\ldots,\mu(t_k)_{q_k})\,.
\end{aligned}
$$

Here the fourth equality follows by the fact that $(f((P_1)_{i_1},\ldots,(P_k)_{i_k}),t)$ is unequal to 0 only if t is of the form $f(t_1,\ldots,t_k)$.

Since both sides of the equation of our theorem coincide, the theorem is proven. □

4 Recognizable tree transducers and recognizable tree series

It is easy to see that our tree transducers do not preserve the recognizability of tree series. (See the example in the last paragraph of page 18 of Gecseg, Steinby [5].) On the other hand, linear root-to-frontier tree transducers do preserve recognizability of tree languages. (See Thatcher [10]; and Gécseg, Steinby [4], Theorem 2.7, Lemma 6.5 and Corollary 6.6.) In this section we show that linear *nondeleting* recognizable tree transducers do preserve recognizability of tree series. We show this by two different constructions: one is based on finite linear systems, the other is based on finite tree automata.

We start with the construction based on finite linear systems.

A *finite linear system* (see Berstel, Reutenauer [1], Bozapalidis [2, 3], Kuich [7, 8]) is a system of formal equations $z_i = p_i$, $1 \le i \le n$, for some $n \ge 1$, where each p_i is in $A\langle\langle T_\Sigma(X \cup Z_n)\rangle\rangle$. A *solution* to the finite linear system $z_i = p_i$, $1 \le i \le n$, is given by $\sigma \in (A\langle\langle T_\Sigma(X)\rangle\rangle)^{n\times 1}$ such that $\sigma_i = p_i(\sigma_1,\ldots,\sigma_n)$, $1 \le i \le n$. A solution σ of $z_i = p_i$, $1 \le i \le n$, is termed *least solution* iff $\sigma \sqsubseteq \tau$ for all solutions τ of $z_i = p_i$, $1 \le i \le n$. The *approximation sequence* $(\sigma^j \mid j \in \mathbb{N})$, $\sigma^j \in (A\langle\langle T_\Sigma(X)\rangle\rangle)^{n\times 1}$, $j \ge 0$, *associated to the finite linear system* $z_i = p_i$, $1 \le i \le n$, is defined as follows:

$$\sigma_i^0 = 0, \qquad \sigma_i^{j+1} = p_i(\sigma_1^j,\ldots,\sigma_n^j),\ 1 \le i \le n,\ j \ge 0\,.$$

The least upper bound $\sigma = \sup(\sigma^j \mid j \in \mathbb{N})$ of the approximation sequence exists and is the least solution of the finite linear system.

A finite linear system $z_i = p_i$, $1 \le i \le n$ is called *proper* iff $(p_i, z_j) = 0$ for all $1 \le j \le n$, i. e., iff there do not appear linear terms in p_i.

A finite linear system $z_i = p_i$, $1 \leq i \leq n$ is called *polynomial* iff each p_i is in $A\langle T_\Sigma(X \cup Z_n)\rangle$. The collection of all the components of least solutions of finite polynomial linear systems is denoted by $A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle$. The tree series in $A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle$ are called *recognizable tree series.*

A finite linear system $z_i = p_i$, $1 \leq i \leq n$ is called *recognizable* iff each p_i is in $A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Z_n)\rangle\rangle$.

An adaption of the proof of Proposition 6.1 of Berstel, Reutenauer [1] yields the following result.

Theorem 8 *For each finite (resp. recognizable finite or polynomial finite) linear system there exists a proper finite (resp. proper recognizable finite or proper polynomial finite) linear system with the same least solution. A proper finite linear system has a unique solution.*

We now show that the least solution of a recognizable finite linear system has recognizable components.

Theorem 9 *Let $z_i = p_i$, $1 \leq i \leq n$, be a recognizable finite linear system with least solution σ. Then $\sigma_i \in A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle$ for all $1 \leq i \leq n$.*

Proof. Without loss of generality let $z_i = p_i$, $1 \leq i \leq n$, be a proper recognizable finite linear system. Since $p_i \in A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Z_n)\rangle\rangle$, $1 \leq i \leq n$, there exist proper polynomial finite linear systems $y_{ij} = q_{ij}$, $1 \leq j \leq m_i$, $m_i \geq 1$, where the y_{ij} are new variables and $q_{ij} \in A\langle T_\Sigma(X \cup Z_n \cup \{y_{i1}, \ldots, y_{im_i}\})\rangle$, $1 \leq i \leq n$, such that the q_{i1}-components of their least solutions τ_i are equal to p_i. Consider now the polynomial finite linear system $z_i = q_{i1}(z_1, \ldots, z_n, y_{i1}, \ldots, y_{im_i})$, $y_{ij} = q_{ij}(z_1, \ldots, z_n, y_{i1}, \ldots, y_{im_i})$, $1 \leq j \leq m_i$, $1 \leq i \leq n$, and observe that this polynomial finite linear system has a unique solution. We claim that this unique solution is given by $\sigma \cup ((\tau_i)_j(\sigma_1, \ldots, \sigma_n) \mid 1 \leq j \leq m_i,\ 1 \leq i \leq n)$. Substitution of this vector yields, for $1 \leq j \leq m_i$, $1 \leq i \leq n$,

$$\begin{aligned} &q_{i1}(\sigma_1, \ldots, \sigma_n, (\tau_i)_1(\sigma_1, \ldots, \sigma_n), \ldots, (\tau_i)_{m_i}(\sigma_1, \ldots, \sigma_n)) = \\ &\qquad (\tau_i)_1(\sigma_1, \ldots, \sigma_n) = p_i(\sigma_1, \ldots, \sigma_n) = \sigma_i\,, \\ &q_{ij}(\sigma_1, \ldots, \sigma_n, (\tau_i)_1(\sigma_1, \ldots, \sigma_n), \ldots, (\tau_i)_{m_i}(\sigma_1, \ldots, \sigma_n)) = (\tau_i)_j(\sigma_1, \ldots, \sigma_n)\,. \end{aligned}$$

Hence $\sigma \cup ((\tau_i)_j(\sigma_1, \ldots, \sigma_n) \mid 1 \leq j \leq m_i,\ 1 \leq i \leq n)$ is the unique solution of the polynomial finite linear system and $\sigma \in (A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle)^{n \times 1}$. □

Let $Y = \{y_i \mid i \geq 1\}$ be an alphabet of new variables and denote $Y_k = \{y_1, \ldots, y_k\}$, $k \geq 1$, $Y_0 = \emptyset$. Let $s(y_1, \ldots, y_k) \in A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle$ and $\tau_j \in A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle$, $1 \leq j \leq k$. Then, by Bozapalidis [2], $s(\tau_1, \ldots, \tau_n)$ is again in $A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle$, i. e., $A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle$ is closed under substitution.

Theorem 10 *$A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle$, $k \geq 1$, is closed under substitution.*

Consider a finite linear system $y_i = p_i(y_1, \ldots, y_n)$, $1 \leq i \leq n$, where $p_i \in A\langle\langle T_\Sigma(X \cup Y_n)\rangle\rangle$, and a tree representation μ with state set Q, where $\mu : \Sigma_k \to (A\langle\langle T_{\Sigma'}(X' \cup Z_k)\rangle\rangle)^{Q \times Q^k}$, $k \geq 1$, and $\mu : \Sigma_0 \cup X \to (A\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q \times 1}$. Let $(y_i)_q$,

$1 \leq i \leq n$, $q \in Q$, be new variables and denote $Y_Q^k = \{(y_i)_q \mid 1 \leq i \leq k,\ q \in Q\}$. Extend the definition of μ to the domain $\Sigma \cup X \cup Y_n$, by

$$\mu : Y_n \to (A\langle\langle T_{\Sigma'}(Y_Q^n)\rangle\rangle)^{Q\times 1},$$

where $\mu(y_j)_q = (y_j)_q$, $1 \leq j \leq n$, $q \in Q$. By this extension, we obtain that

$$\mu : T_\Sigma(X \cup Y_n) \to (A\langle\langle T_{\Sigma'}(X' \cup Y_Q^n)\rangle\rangle)^{Q\times 1}.$$

Lemma 11 *Consider $s(y_1, \ldots, y_n) \in A\langle\langle T_\Sigma(X \cup Y_n)\rangle\rangle$ and a linear nondeleting tree representation μ with domain $\Sigma \cup X \cup Y_n$. Let $s_1, \ldots, s_n \in A\langle\langle T_\Sigma(X)\rangle\rangle$. Then*

$$\mu(s)[\mu(s_j)_q/(y_j)_q,\ 1 \leq j \leq n,\ q \in Q] = \mu(s(s_1, \ldots, s_n)).$$

Proof. We first consider a tree $t \in T_\Sigma(X \cup Y_n)$ and show by induction on the form of t that $\mu(t)[\mu(s_j)_q/(y_j)_q,\ 1 \leq j \leq n,\ q \in Q] = \mu(t(s_1, \ldots, s_n))$.
(i) For $t = y_i$, $1 \leq i \leq n$, we obtain $\mu(y_i)[\mu(s_j)/\mu(y_j),\ 1 \leq j \leq n] = \mu(s_i) = \mu(y_i(s_1, \ldots, s_n))$.
(ii) For $t = x$, $x \in \Sigma_0 \cup X$, we obtain $\mu(x)[\mu(s_j)/\mu(y_j),\ 1 \leq j \leq n] = \mu(x) = \mu(x(s_1, \ldots, s_n))$.
(iii) For $t = f(t_1, \ldots, t_k)$, $f \in \Sigma_k$, $t_1, \ldots, t_k \in T_\Sigma(X \cup Y_n)$, $k \geq 1$, we obtain

$$\begin{aligned}
&\mu(f(t_1, \ldots, t_k))[\mu(s_j)/\mu(y_j),\ 1 \leq j \leq n] = \\
&\mu(f)(\mu(t_1)[\mu(s_j)/\mu(y_j),\ 1 \leq j \leq n], \ldots, \mu(t_k)[\mu(s_j)/\mu(y_j),\ 1 \leq j \leq n]) = \\
&\mu(f)(\mu(t_1(s_1, \ldots, s_n)), \ldots, \mu(t_k(s_1, \ldots, s_n))) = \\
&\mu(f(t_1(s_1, \ldots, s_n), \ldots, t_k(s_1, \ldots, s_n))) = \\
&\mu((f(t_1, \ldots, t_k))(s_1, \ldots, s_n)).
\end{aligned}$$

Here we have applied the induction hypothesis in the second equality and Theorem 7 in the third equality.

Finally, we obtain

$$\begin{aligned}
&\mu(s)[\mu(s_j)/\mu(y_j),\ 1 \leq j \leq n] = \\
&\textstyle\sum_{t\in T_\Sigma(X\cup Y_n)}(s,t) \otimes \mu(t)[\mu(s_j)/\mu(y_j),\ 1 \leq j \leq n] = \\
&\textstyle\sum_{t\in T_\Sigma(X\cup Y_n)}(s,t) \otimes \mu(t(s_1, \ldots, s_n)) = \\
&\mu(\textstyle\sum_{t\in T_\Sigma(X\cup Y_n)}(s,t)t(s_1, \ldots, s_n)) = \mu(s(s_1, \ldots, s_n)).
\end{aligned}$$

□

Theorem 12 *Consider a linear nondeleting tree representation μ with domain $\Sigma \cup X \cup Y_n$. Let $y_i = p_i(y_1, \ldots, y_n)$, $1 \leq i \leq n$, where $p_i \in A\langle\langle T_\Sigma(X \cup Y_n)\rangle\rangle$, be a finite linear system with least solution σ. Then $\mu(\sigma)$ is the least solution of the finite linear system $\mu(y_i) = \mu(p_i(y_1, \ldots, y_n))$, $1 \leq i \leq n$.*

Proof. Let $(\sigma^j \mid j \in \mathbb{N})$ and $(\tau^j \mid j \in \mathbb{N})$ be the approximation sequences of $y_i = p_i(y_1, \ldots, y_n)$, $1 \leq i \leq n$, and $\mu(y_i) = \mu(p_i(y_1, \ldots, y_n))$, $1 \leq i \leq n$,

respectively. We claim that $\tau_i^j = \mu(\sigma_i^j)$, $1 \leq i \leq n$, $j \geq 0$, and show it by induction on j. The case $j = 0$ is clear. Let $j \geq 0$. Then, for $1 \leq i \leq n$,

$$\begin{aligned}
&\tau_i^{j+1} = \mu(p_i(y_1, \ldots, y_n))[\tau_k^j/\mu(y_k),\ 1 \leq k \leq n] = \\
&\mu(p_i(y_1, \ldots, y_n))[\mu(\sigma_k^j)/\mu(y_k),\ 1 \leq k \leq n] = \\
&\mu(p_i(\sigma_1^j, \ldots, \sigma_n^j)) = \mu(\sigma_i^{j+1})\,.
\end{aligned}$$

Here we have applied the induction hypothesis in the second equality and Lemma 11 in the third equality. The claim now implies our theorem. □

A tree representation μ is called *recognizable* iff $\mu(f) \in (A^{\mathrm{rec}}\langle\langle T_{\Sigma'}(X' \cup Z_k)\rangle\rangle)^{Q \times Q^k}$ for $f \in \Sigma_k$, $k \geq 1$, and $\mu(f) \in (A^{\mathrm{rec}}\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q \times 1}$ for $f \in \Sigma_0 \cup X$. A tree transducer $\mathfrak{T} = (Q, \mu, S)$ is called *recognizable* iff μ is a recognizable tree representation and the entries of S are of the form $S_q = a_q z_1$, $a_q \in A$, $q \in Q$.

Corollary 13 *Consider a linear nondeleting recognizable tree representation μ. Let s be in $A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle$. Then $\mu(s)$ is in $(A^{\mathrm{rec}}\langle\langle T_{\Sigma'}(X')\rangle\rangle)^{Q \times 1}$.*

Corollary 14 *Consider a linear nondeleting recognizable tree transducer $\mathfrak{T}$ and a recognizable tree series s. Then $||\mathfrak{T}||(s)$ is again recognizable.*

We now turn to the automata-based construction.

Our tree automata are a generalization of the nondeterministic root-to-frontier tree recognizers (see Gécseg, Steinby [4, 5]) and are defined in Kuich [7, 8]. A *tree automaton* (with input alphabet Σ and leaf alphabet X)

$$\mathfrak{A} = (I, M, S, P)$$

is given by

(i) a nonempty set I of *states*,

(ii) a sequence $M = (M_k \mid k \geq 1)$ of *transition matrices* $M_k \in (A\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle)^{I \times I^k}$, $k \geq 1$,

(iii) $S \in (A\langle\langle T_\Sigma(X \cup Y_1)\rangle\rangle)^{1 \times I}$, called the *initial state vector*,

(iv) $P \in (A\langle\langle T_\Sigma(X)\rangle\rangle)^{I \times 1}$, called the *final state vector*.

The *approximation sequence* $(\sigma^j \mid j \in \mathbb{N})$, $\sigma^j \in (A\langle\langle T_\Sigma(X)\rangle\rangle)^{I \times 1}$, $j \geq 0$, *associated to* $\mathfrak{A}$ is defined as follows:

$$\sigma^0 = 0, \qquad \sigma^{j+1} = \sum_{k \geq 1} M_k(\sigma^j, \ldots, \sigma^j) + P, \quad j \geq 0\,.$$

The *behavior* $||\mathfrak{A}|| \in A\langle\langle T_\Sigma(X)\rangle\rangle$ of the tree automaton $\mathfrak{A}$ is defined by

$$||\mathfrak{A}|| = \sum_{i \in I} S_i(\sigma_i) = S(\sigma)\,,$$

where $\sigma \in (A\langle\langle T_\Sigma(X)\rangle\rangle)^{I\times 1}$ is the least upper bound of the approximation sequence associated to $\mathfrak{A}$.

A tree automaton $\mathfrak{A} = (I, (M_k \mid k \geq 1), S, P)$ is termed *simple* iff the entries of the transition matrices M_k, $k \geq 1$, of the initial state vector S and of the final state vector P have the following specific form:

(i) the entries of M_k, $k \geq 2$, are of the form $\sum_{f\in\Sigma_k} a_f f(y_1, \ldots, y_k)$, $a_f \in A$;

(ii) the entries of M_1 are of the form $\sum_{f\in\Sigma_1} a_f f(y_1) + ay_1$, $a_f, a \in A$;

(iii) the entries of P are of the form $\sum_{\omega\in\Sigma_0\cup X} a_\omega \omega$, $a_\omega \in A$;

(iv) the entries of S are of the form dy_1, $d \in A$.

A tree automaton $\mathfrak{A} = (I, (M_k \mid k \geq 1), S, P)$ is termed *proper* iff the entries of M_1 do not contain a linear term ay_1, $a \in A$.

A tree automaton $\mathfrak{A} = (I, M, S, P)$ is called *polynomial* (resp. *recognizable*) iff the following conditions are satisfied:

(i) $M = (M_k \mid 1 \leq k \leq \bar{k})$ is a *finite sequence* of transition matrices M_k whose entries are *polynomials* in $A\langle T_\Sigma(X \cup Y_k)\rangle$ (resp. *tree series* in $A^{\mathrm{rec}}\langle\langle T_\Sigma(X \cup Y_k)\rangle\rangle$), $1 \leq k \leq \bar{k}$. (Technically speaking, this means that all transition matrices $M_{\bar{k}+j}$, $j \geq 1$, are equal to the zero matrix.) Moreover, the matrices M_k, $1 \leq k \leq \bar{k}$, are *row finite*.

(ii) The entries of the initial state vector S are of the form $S_i = d_i y_1$, $i \in I$. Moreover, S is *row finite*.

(iii) The entries of the final state vector P are *polynomials* in $A\langle T_\Sigma(X)\rangle$ (resp. *recognizable tree series* in $A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle$).

By Bozapalidis [2], by Kuich [7], and by Theorem 9 we obtain the following result.

Theorem 15 *The following statements on a formal tree series in* $A\langle\langle T_\Sigma(X)\rangle\rangle$ *are equivalent:*

(i) $s \in A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle$,

(ii) *there exists a polynomial tree automaton* $\mathfrak{A}$ *with finite state set such that* $s = ||\mathfrak{A}||$,

(iii) *there exists a simple proper polynomial tree automaton* $\mathfrak{A}$ *with finite state set such that* $s = ||\mathfrak{A}||$,

(iv) *there exists a recognizable tree automaton* $\mathfrak{A}$ *with finite state set such that* $s = ||\mathfrak{A}||$,

(v) *there exists a proper recognizable tree automaton* $\mathfrak{A}$ *with finite state set such that* $s = ||\mathfrak{A}||$,

Let $\mathfrak{A} = (I, (M_k \mid k \geq 1), S, P)$ be a simple tree automaton and $\mathfrak{T} = (Q, \mu, R)$ be a tree transducer such that $R_q = a_q y_1$, $a_q \in A$, $q \in Q$. Then $\mathfrak{T}(\mathfrak{A})$ is defined to be the tree automaton

$$\mathfrak{T}(\mathfrak{A}) = (I \times Q, (\mu(M_k) \mid k \geq 1), S \otimes R, \mu(P)) .$$

Theorem 16 *Let $\mathfrak{A} = (I, (M_k \mid k \geq 1), S, P)$ be a simple tree automaton and $\mathfrak{T} = (Q, \mu, R)$ be a linear nondeleting tree transducer. Then*

$$||\mathfrak{T}(\mathfrak{A})|| = ||\mathfrak{T}||(||\mathfrak{A}||) .$$

Proof. Consider the approximation sequences $(\sigma^j \mid j \in \mathbb{N})$ and $(\tau^j \mid j \in \mathbb{N})$ of $\mathfrak{A}$ and $\mathfrak{T}(\mathfrak{A})$ with upper bounds σ and τ, respectively. Then we prove by inducion on j that $\tau^j = \mu(\sigma^j)$, $j \geq 0$. The induction basis being clear, we proceed with the induction step. Let $j \geq 0$. Then

$$\begin{array}{l} \tau^{j+1} = \sum_{k\geq 1} \mu(M_k)(\mu(\sigma^j), \ldots, \mu(\sigma^j)) + \mu(P) = \\ \sum_{k\geq 1} \mu(M_k(\sigma^j, \ldots, \sigma^j)) + \mu(P) = \\ \mu(\sum_{k\geq 1} M_k(\sigma^j, \ldots, \sigma^j) + P) = \mu(\sigma^{j+1}) . \end{array}$$

Here the second equality follows by Theorem 7. Hence, we obtain $\tau = \mu(\sigma)$.

We now compute the behavior of $\mathfrak{T}(\mathfrak{A})$:

$$\begin{array}{l} ||\mathfrak{T}(\mathfrak{A})|| = (S \otimes R)(\tau) = \sum_{i\in I} \sum_{q\in Q} ((S \otimes R)_i)_q (\mu(\sigma)_i)_q = \\ \sum_{q\in Q} \sum_{i\in I} R_q S_i \sum_{t\in T_\Sigma(X)} (\sigma_i, t)\mu(t)_q = \\ \sum_{q\in Q} R_q \sum_{t\in T_\Sigma(X)} \sum_{i\in I} (S_i \sigma_i, t)\mu(t)_q = \\ \sum_{q\in Q} R_q \sum_{t\in T_\Sigma(X)} (||\mathfrak{A}||, t)\mu(t)_q = \\ \sum_{q\in Q} R_q \mu(||\mathfrak{A}||)_q = ||\mathfrak{T}||(||\mathfrak{A}||) . \end{array}$$

□

Corollary 17 *Let $\mathfrak{A}$ be a simple polynomial tree automaton with finite state set and $\mathfrak{T}$ be a linear nondeleting recognizable tree transducer. Then $||\mathfrak{T}||(||\mathfrak{A}||)$ is in $A^{\mathrm{rec}}\langle\langle T_\Sigma(X)\rangle\rangle$.*

Acknowledgement. Many thanks are due to Ferenc Gécseg for discussions on root-to-frontier tree transducers.

References

[1] Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theor. Comput. Sci. 18(1982) 115–148.

[2] Bozapalidis, S.: Equational elements in additive algebras. Technical Report, Aristotle University of Thessaloniki, 1997.

[3] Bozapalidis, S.: Context-free series on trees. Preprint, Aristotle University of Thessaloniki, February 1998.

[4] Gécseg, F., Steinby, M.: Tree Automata. Akademiai Kiado, 1984.

[5] Gécseg, F., Steinby, M.: Tree Languages. In: Handbook of Formal Languages (Eds.: G. Rozenberg and A. Salomaa), Springer, 1997, Vol. 3, Chapter 1, 1–68.

[6] Kuich, W.: Semirings and formal power series: Their relevance to formal languages and automata theory. In: Handbook of Formal Languages (Eds.: G. Rozenberg and A. Salomaa), Springer, 1997, Vol. 1, Chapter 9, 609–677.

[7] Kuich, W.: Formal power series over trees. In: Proceedings of the 3rd International Conference Developments in Language Theory (S. Bozapalidis, ed.), Aristotle University of Thessaloniki, 1998, 61–101.

[8] Kuich, W.: Pushdown tree automata, algebraic tree systems, and algebraic tree series. Preprint, Technische Universität Wien, 1998.

[9] Kuich, W., Salomaa, A.: Semirings, Automata, Languages. EATCS Monographs on Theoretical Computer Science, Vol. 5. Springer, 1986.

[10] Thatcher, J. W.: Generalized^2 sequential machine maps. IBM Research Report RC 2466, 1969.

Acta Cybernetica **14** (1999) 151-164.

Duplication Grammars *

Carlos MARTÍN-VIDE † Gheorghe PĂUN ‡

Abstract

Motivated by the abundance of duplication operations appearing in natural languages and in the genetic area, we introduce a generative mechanism based on duplication operations: one starts from a given finite set of strings and one produces new strings by copying certain substrings, according to a set of given rules (which specify contexts where duplicated substrings can be inserted). We mainly investigate the power of such devices, comparing the obtained families of languages to each other and with families in the Chomsky hierarchy. In this context, we also solve a problem left open in a paper by Dassow and Mitrana, [1], even giving a stronger results: the iterated duplication with rules of only one type (see formal definitions in the sequel) can generate non-context-free languages (even non-matrix languages).

1 Introduction

The duplication of strings or of parts of strings is a common operation both in natural languages and in the genetic languages. We refer to [12], [15] for discussions about this topic from the linguistical point of view; it is interesting to note that it seems that reduplication, which is a non-context-free type of operation, is more abundant in natural languages than mirror-image constructions, which can be handled by linear Chomsky grammars. This is an argument supporting the thesis that Chomsky grammars and their classification is not an adequate model of the syntax of natural languages. Details about operations appearing in the genetic area, duplication included, can be found, for instance, in [2], [10], [11], [18], or in the first chapter of [14].

We start here from the approach in [1] to duplication in the DNA area. In that paper, one considers *duplication rules* of the form (u_1, v, u_2), where u_1, v, u_2 are strings (over the DNA alphabet, but the operation can be defined in general, over any finite alphabet); the idea is that the string v can be inserted in between the strings u_1, u_2, providing that it already appears in the processed string. This

*Research supported by the Direcció General de Recerca, Generalitat de Catalunya (PIV)

†Research Group in Mathematical Linguistics and Language Engineering Rovira i Virgili University Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain e-mail: cmv@astor.urv.es

‡Institute of Mathematics of the Romanian Academy PO Box 1 - 764, 70700 Bucureşti, Romania, e-mail: gpaun@imar.ro.

amounts to a duplication of v. Both the one-step operation of this type and the iterated operation are considered in [1], where one proves that the non-iterated operation preserves the regular and the context-free languages. It is stated as an open problem in [1] the question whether or not the iterated duplication preserves the context-freeness.

We distinguish here three types of duplications: taking v from the left of the place where a copy of it will be produced, from the right of that place, or duplicating v near an already existing copy of it; in all cases, the context (u_1, u_2) controls the operation. We prove that in each of these cases, the iterated duplication can lead finite languages to languages which cannot be generated by matrix languages (without appearance checking). This solves the problem in [1], in a stronger form. The result is not surprising, because of the context-sensitivity of the operations we use (the presence of the context (u_1, u_2) associated with each string v which can be duplicated); this also corresponds to the situation met for the so-called *insertion grammars* of [5], where the string to be inserted is not necessarily a substring of the current string (see the corresponding chapter in [13]). Somewhat unexpected is the fact that non-context-free languages can be also produced when starting from finite languages and using duplication operations in the restricted case when the copy of the duplicated string is produced adjacent to the string itself.

2 Formal Language Theory Prerequisites

We introduce a few notions and notations necessary in the sequel; for details, the reader is referred to [16].

For an alphabet V, we denote by V^* the set of all strings over V, including the empty one, denoted by λ; the set of non-empty strings over V is denoted by V^+. The length of $x \in V^*$ is denoted by $|x|$ and the number of occurrences of a symbol $a \in V$ in a string $x \in V^*$ is denoted by $|x|_a$. The left derivative of a language $L \subseteq V^*$ with respect to a string $x \in V^*$ is $\partial_x^l(L) = \{w \in V^* \mid xw \in L\}$ and the right derivative is $\partial_x^r(L) = \{w \in V^* \mid wx \in L\}$.

A *context-free grammar* is a construct $G = (N, T, S, P)$, where N is the non-terminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom, and P is the set of production rules, pairs of the form (A, x) with $A \in N, x \in (N \cup T)^*$ (written in the form $A \to x$). For $x, y \in (N \cup T)^*$, we write $x \Longrightarrow y$ if and only if $x = x_1 A x_2, y = x_1 z x_2$, for some $x_1, x_2 \in (N \cup T)^*$ and $A \to z \in P$. By $\Longrightarrow^*$ we denote the reflexive and transitive closure of the relation $\Longrightarrow$. The language generated by G is defined by $L(G) = \{x \in T^* \mid S \Longrightarrow^* x\}$.

By *FIN, REG, LIN, CF, CS, RE* we denote the families of finite, regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively. This is the *Chomsky hierarchy*, the standard test bed for any new type of language generating devices.

3 Duplication Grammars

We now introduce the main object of our study, in several variants.

A *duplication grammar* is a construct

$$\Delta = (V, D_l, D_r, D_0, A),$$

where V is an alphabet, D_l, D_r, D_0 are finite subsets of $V^* \times V^+ \times V^*$, and A is a finite language over V.

The elements of the sets D_l, D_r, D_0 are called *duplication rules*, those of A are called *axioms*. Note that the duplication rules have the second element non-empty.

With respect to such a grammar, for $x, y \in V^*$ we define:

$$\begin{aligned}
x \Longrightarrow_{D_l} y \quad &\text{iff} \quad x = x_1u_1u_2x_2vx_3, y = x_1u_1vu_2x_2vx_3, \\
&\quad \text{for some } x_1, x_2, x_3 \in V^*, (u_1, v, u_2) \in D_l, \\
x \Longrightarrow_{D_r} y \quad &\text{iff} \quad x = x_1vx_2u_1u_2x_3, y = x_1vx_2u_1vu_2x_3, \\
&\quad \text{for some } x_1, x_2, x_3 \in V^*, (u_1, v, u_2) \in D_r, \\
x \Longrightarrow_{D_0} y \quad &\text{iff} \quad x = x_1u_1vu_2x_2, y = x_1u_1vvu_2x_2, \\
&\quad \text{for some } x_1, x_2 \in V^*, (u_1, v, u_2) \in D_0.
\end{aligned}$$

We write $\Longrightarrow$ for denoting any of these relations $\Longrightarrow_\alpha$; the reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$. Then, the *language generated* by the grammar Δ is defined by

$$L(\Delta) = \{w \in V^* \mid z \Longrightarrow^* w, \text{ for some } z \in A\}.$$

In words, one starts from strings in A and one iteratively duplicates subtrings as allowed by the triples in the sets D_l, D_r, D_0. In all cases, a copy of a substring of the current string is produced, to the left of its former occurrence, to the right, or adjacent to that occurrence, respectively.

Because one or two of the three sets D_l, D_r, D_0 can be empty, we obtain in this way seven classes of grammars and, correspondingly, seven families of languages. We denote by $DUPL(\alpha)$ the family of languages generated by duplication grammars containing rules of the type α, where α can be one of $l, r, 0, lr, l0, r0, lr0$; the absence of one of the symbols $l, r, 0$ means that the corresponding sets of duplication rules is empty.

Several duplication grammars and languages generated by them will be considered in the following sections, hence we do not give here examples.

4 Generative Capacity

We compare the families $DUPL(\alpha)$ to each other and to families of languages in the Chomsky hierarchy.

Note that each duplication language has the bounded growth property: for each infinite language L there is a constant k such that if $x \in L$, then there is $y \in L, y \neq x$, with $||x| - |y|| \leq k$.

Directly from the definitions, we obtain the relations in the diagram in Figure 1 (an arrow from a lower family to an upper family indicates an inclusion which is not necessarily proper). This diagram will be useful below when investigating the relationships between these families.

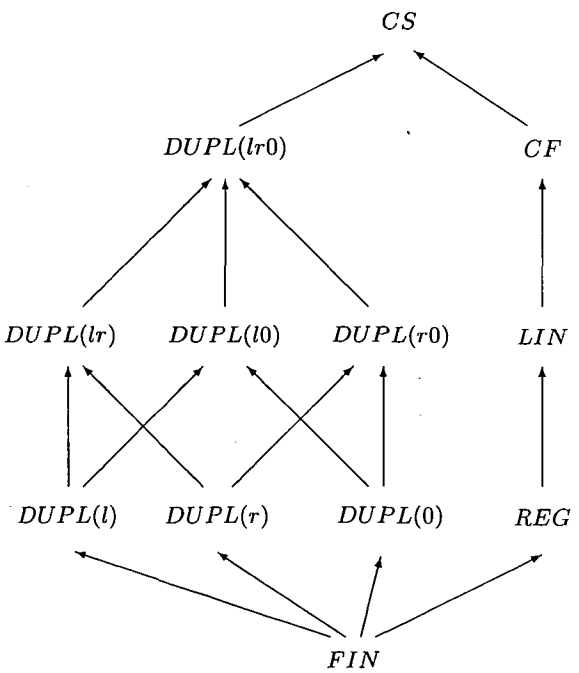


Figure 1

Lemma 1. *Let V be an alphabet containing at least three symbols. The language V^+ is in $DUPL(l) \cap DUPL(r)$, but not in $DUPL(0)$.*

Proof. For the duplication grammars $\Delta = (V, D_l, D_r, \emptyset, A)$, where one of D_l, D_r is empty and the other is equal to $\{(\lambda, a, \lambda) \mid a \in V\}$, and

$$A = \{x \in V^+ \mid |x|_a \leq 1 \text{ for each } a \in V\},$$

we obviously have $L(\Delta) = V^+$.

The language V^+ cannot be generated by a duplication grammar which uses only rules in D_0, because at each step of the form $x \Longrightarrow_{D_0} y$ we produce a string y

of the form $y = y_1vvy_2$ for some $v \in V^+$. However, for V containing at least three symbols, there are arbitrarily long square-free strings in V^+, [19], [17]. □

Remark 1. (Suggested to us, without a proof, by V. Mitrana) The language V^+ for $V = \{a, b\}$ is in $DUPL(0)$, that is, the previous result cannot be improved by considering V with only two symbols. Indeed, the duplication grammar

$$\Delta = (\{a, b\}, \emptyset, \emptyset, D_0, A),$$
$$A = \{a, b, ab, aba, ba, bab\},$$
$$D_0 = \{(\lambda, a, \lambda),\ (\lambda, b, \lambda),\ (\lambda, ab, \lambda),\ (\lambda, ba, \lambda)\},$$

generates V^+. The inclusion $L(\Delta) \subseteq V^+$ is obvious, the converse inclusion can be proved as follows. For a string $x = c_1^{i_1}c_2^{i_2}\dots c_r^{i_r}, r \geq 1, c_j \in \{a, b\}, i_j \geq 1$, for all $1 \leq j \leq r$, with $c_j \neq c_{j+1}$, $1 \leq j \leq r-1$, we denote by $red(x)$ the (*reduced*) string $c_1c_2\dots c_r$ (all blocks of symbols a and b are reduced to one symbol). Clearly, starting from strings in A and using the rules $(\lambda, ab, \lambda), (\lambda, ba, \lambda)$, we can generate all strings $w \in V^+, w = red(x)$, for some $x \in V^+$. Then, by using rules $(\lambda, a, \lambda), (\lambda, b, \lambda)$, we can pass from $red(x)$ to x, for any $x \in V^+$.

It is easy to see that $\{a, b\}^+ \in DUPL(l) \cap DUPL(r)$, too, and that $a^+ \in DUPL(\alpha)$ for all $\alpha \in \{l, r, 0\}$.

Lemma 2. (i) $\{aba^nb^n \mid n \geq 1\} \in DUPL(r) - DUPL(l)$. (ii) $\{a^nb^nab \mid n \geq 1\} \in DUPL(l) - DUPL(r)$.

Proof. (i) The duplication grammar

$$\Delta = (\{a, b\}, \emptyset, \{(a, ab, b)\}, \emptyset, \{abab\})$$

obviously generates the language $\{aba^nb^n \mid n \geq 1\}$.

This language cannot be generated by a duplication grammar which uses only rules in D_l: in order to produce strings aba^nb^n with arbitrarily large n we need to use triples of the form $(a^j, a^ib^i, b^k) \in D_l$; such a triple can be used for introducing a^ib^i in only one position of a string aba^mb^m, namely in between a^m and b^m; there is no occurrence of a^ib^i to the right of that position, as requested by the definition of the relation $\Longrightarrow_{D_l}$.

Assertion (ii) can be proved in the same way. □

Lemma 3. $\{a^nb^n \mid n \geq 1\} \notin DUPL(lr0)$.

Proof. No derivation step is possible starting from a string of the form a^nb^n because no substring of such a string can be duplicated. □

In [1] one asks whether or not the context-freeness is preserved by the iterated duplication of types $\Longrightarrow_{D_l}, \Longrightarrow_{D_r}$ (in fact, in [1] one uses only one set D of rules, applied both "to the left" and to "to the right", in the sense of D_l, D_r, respectively). We prove below that the answer is negative: even finite languages are led to non-context-free languages by iteratively duplicating substrings of them. This is one of the main results of our paper.

Lemma 4. $DUPL(0) - CF \neq \emptyset$.

Proof. Let us consider the duplication grammar

$$\Delta = (\{a, b, c\}, \emptyset, \emptyset, D_0, A),$$

where D_0 contains the following rules

$$\begin{aligned} r_1 &= (ca, ab, baabbaabb), \\ r_2 &= (aababba, ab, b), \\ r_3 &= (caa, b, abbaabab), \\ r_4 &= (caabb, a, bbaabab), \\ r_5 &= (bbaabb, a, bbaabab), \\ r_6 &= (aabbaa, b, abbaabab), \\ r_7 &= (aabbaa, b, abbc), \\ r_8 &= (bbaabb, a, bbc), \end{aligned}$$

and

$$A = \{caabbaabbaabbc\}.$$

Consider also the regular language

$$R = c(aabb)^+c,$$

and examine the intersection $L(\Delta) \cap R$.

Starting from a string of the form $c(aabb)^nc$ (initially, we have $n = 3$), the rules r_1, r_2 double the substrings ab, from left to right:

$$\begin{aligned} & caabbaabbaabb \ldots aabbc \\ \Longrightarrow_{D_0} \; & caababbaabbaabb \ldots aabbc \\ \Longrightarrow_{D_0} \; & caababbaababbaabb \ldots aabbc \\ & \ldots\ldots\ldots\ldots\ldots \\ \Longrightarrow_{D_0} \; & caababbaababbaababb \ldots aababbc. \end{aligned}$$

The rules $r_3 - r_8$ can be applied to a string obtained in this way in order to double all occurrences of a and b which are not double. This is also done from left to right:

$$\begin{aligned} & caababbaababbaababb \ldots aababbc \\ \Longrightarrow_{D_0} \; & caabbabbaababbaababb \ldots aababbc \\ \Longrightarrow_{D_0} \; & caabbaabbaababbaababb \ldots aababbc \\ & \ldots\ldots\ldots\ldots\ldots \\ \Longrightarrow_{D_0} \; & caabbaabbaabbaabbaabbaabb \ldots aabbaabbc. \end{aligned}$$

In this way, each substring $aabb$ is doubled: we pass from $aabb$ to $aababb$ and then to $aabbaabb$. Thus, the obtained string is $c(aabb)^{2n}c$. The process can be repeated.

In order to obtain a string in R, the operations of doubling the substrings ab and then of doubling the symbols a, b which are not appearing in substrings aa, bb, respectively, must be completed. Indeed, consider the case of a string of the form

$$w = caababb \ldots aa\underline{ab}abbaababbaabbaabb \ldots aabbc,$$

that is, obtained after some steps where rules r_1, r_2 were applied. Start now to use the other rules. The rules r_3, r_4, r_5, r_6 can be used from the left to the right and the symbols a, b not appearing in blocks aa, bb are doubled. Assume that we perform this operation the maximal possible number of times, that is we double also the underlined symbols in the string w; we obtain the string

$$w' = caabbaabb \ldots aabbaabbaababbaabbaabb \ldots aabbc.$$

No further application of rules r_5, r_6 is possible, hence no further occurrence of a, b can be doubled. We have first to continue with the rule r_2, doubling new occurrences of ab and making possible the identification of the right contexts of rules r_5, r_6 in the current string.

Symmetrically, consider a string obtained after some steps where symbols a, b were doubled:

$$z = caabbaabb \ldots aabbaabbaababbaababb \ldots aababbc.$$

From the left, we can start doubling substrings ab; this must be done step by step, but cannot proceed ahead of the doubling of the symbols a, b. For instance, we can obtain the string

$$z' = caababbaababb \ldots aababbaababba\underline{ab}abbaababb \ldots aababbc,$$

but we cannot go further, with the underlined substring ab. Again we have to continue with the previous operation of doubling (now, that of doubling the symbols a and b appearing separately).

Consequently, in order to get a string in R we have to perform complete "translations" of the string, that is doublings of the number of occurrences of the blocks $aabb$. This implies the equality

$$L(\Delta) \cap R = \{c(aabb)^{3 \cdot 2^n} c \mid n \geq 0\}.$$

Clearly, this is not a context-free language, hence $L(\Delta)$ is non-context-free either. □

The family of languages generated by matrix (programmed, controlled, etc) grammars with context-free rules (without using appearance checking) is closed under intersection with regular languages and under morphisms, [3]. Moreover, each one-letter matrix language is regular, [9]. This proves that the language $L(\Delta)$ in the previous proof is not a matrix one (with the morphism h defined by $h(a) = h(b) = h(c) = a$ we obtain $h(L(\Delta) \cap R) = \{a^{12 \cdot 2^n + 2} \mid n \geq 0\}$, which is not regular). Thus, we can conclude that $DUPL(0)$ contains non-matrix languages.

Corollary 1. $DUPL(l) - CF \neq \emptyset$, $DUPL(r) - CF \neq \emptyset$.

Proof. Let $\Delta = (\{a,b,c\}, \emptyset, \emptyset, D_0, \{caabbaabbaabbc\})$ be the duplication grammar constructed in the previous proof and consider the following grammar:

$$\Delta_r = (\{a,b,c\}, \emptyset, D_r, \emptyset, \{abcaabbaabbaabbc\}),$$
$$D_r = \{(u_1 v, v, u_2) \mid (u_1, v, u_2) \in D_0\}.$$

For the regular language $R_r = abc(aabb)^+c$ we obtain

$$L(\Delta_r) \cap R_r = \{ab\}(L(\Delta) \cap R).$$

(The rules in the set D_r of Δ_r lead to duplications identical to those controlled by the rules in the set D_0 of Δ; the string ab in the left end of the strings provides the necessary strings a, b, ab for duplications.) This proves that the language $L(\Delta_r)$ is not context-free.

A similar modification of Δ leads to a grammar Δ_l (with only the set D_l non-empty) generating a non-context-free language (we take the axiom $c(aabb)^3cab$ and $R_l = c(aabb)^+cab$). □

Clearly, if we allow the use of each triple (u_1, v, u_2) in the above grammars Δ_l, Δ_r in any mode $\Longrightarrow_{D_r}, \Longrightarrow_{D_l}$, then the generated language is not modified, hence the problem in [1] is answered: the iterated duplication does not preserve the context-freeness.

We combine these results in a synthesis theorem:

Theorem 1. (i) *The families* $DUPL(l), DUPL(r)$ *are incomparable. The families* LIN, CF *are incomparable with all families* $DUPL(\alpha), \alpha \in \{l, r, 0, lr, l0, r0, lr0\}$; *REG is incomparable with* $DUPL(0)$.

(ii) *All families* $DUPL(\alpha), \alpha \in \{l, r, 0, lr, l0, r0, lr0\}$, *are strictly included in* CS.

(iii) *The inclusions* $DUPL(l) \subset DUPL(lr), DUPL(r) \subset DUPL(lr), DUPL(0) \subset DUPL(l0), DUPL(0) \subset DUPL(r0)$ *are proper.*

It remains as an *open problem* to clarify the relationships between the families $DUPL(0), REG$ and the families $DUPL(\alpha)$ with $\alpha \neq 0$. Is REG included in $DUPL(\alpha)$? The next result provides a partial answer to this problem.

Theorem 2. *For every regular language* L *there is a string* w *such that* $\{w\}L \in DUPL(r)$ *and* $L\{w\} \in DUPL(l)$.

Proof. Let L be a regular language and let $M = (K, V, q_0, F, \delta)$ be the minimal deterministic finite automaton recognizing L (K is the set of states, V is the alphabet, q_0 is the initial state, F is the set of final states, and $\delta : K \times V \longrightarrow K$ is the transition mapping).

For each $x \in V^*$, we define the mapping $\rho_x : K \longrightarrow K$ by

$$\rho_x(q) = q' \text{ iff } \delta(q, x) = q', \ q \in K.$$

Obviously, if $x_1, x_2 \in V^*$ are such that $\rho_{x_1} = \rho_{x_2}$, then for every $u, v \in V^*$, ux_1v is in L if and only if ux_2v is in L.

The set of mappings from K to K is finite. Hence the set of mappings ρ_x as above is finite. Let n_0 be their number. We construct the duplication grammar $\Delta_r = (V \cup \{c, d\}, \emptyset, D_r, \emptyset, A)$, where c, d are two symbols not in V,

$$D_r = \{(zu_1, v, \lambda) \mid \rho_{u_1} = \rho_{u_1 v}, \text{ for } u_1, v \in V^*, |u_1|, |v| \leq n_0, \text{ and either } z = dz', z' \in V^*, |z'u_1| \leq n_0, \text{ or } z \in V^*, |zu_1| = n_0 + 1\},$$

and A is constructed as follows. Take all strings $x \in V^*$ of length at most n_0, concatenate each of them with c at both ends, then concatenate all the obtained strings, in any given order; denote by w' the obtained string and consider $w = w'd$. Then,

$$A = \{wx \mid x \in L, |x| \leq n_0 + 1\}.$$

Therefore, the string w provides substrings v for duplication, as requested by the rules in D_r. These rules cannot be applied to the substrings of w, because of the presence of symbols c: the left context of each rule in D_r is of a length greater than n_0, or it begins with d, hence this context cannot be found in w.

From the definition of mappings ρ_x and the definitions of A and D_r, it follows immediately that $L(\Delta) \subseteq \{w\}L$.

Assume that the converse inclusion is not true and let $wu \in \{w\}L - L(\Delta)$ be a string of minimal length with this property. Thus $wu \notin A$. Hence $|u| \geq n_0 + 2$. Let $u = zz'$ with $|z'| = n_0$ and $z \in V^*$. If $z' = a_1 a_2 \dots a_{n_0}$, then it has $n_0 + 1$ prefixes, namely $\lambda, a_1, a_1 a_2, \dots, a_1 \dots a_{n_0}$. There are only n_0 different mappings ρ_x. Therefore there are two prefixes u_1, u_2 of z' such that $u_1 \neq u_2$ and $\rho_{u_1} = \rho_{u_2}$. With no loss in generality we may assume that $|u_1| < |u_2|$. By substituting u_2 by u_1 we obtain a string u' which is also in L. As $|u'| < |u|$ and wu was of minimal length in $\{w\}L - L(\Delta)$, we obtain $wu' \in L(\Delta)$. However, $|u_2| - |u_1| \leq |u_2| \leq n_0$, so if $u_2 = u_1 v$, then $(z_1 u_1, v, \lambda) \in D_r$, where either $z_1 = z$ and begins by d, or z_1 is a suffix of z such that $|zu_1| = n_0 + 1$. This implies that $wu' \Longrightarrow_{D_r} wu$, that is $wu \in L(\Delta)$, a contradiction. In conclusion, $\{w\}L \subseteq L(\Delta)$.

In the same way we can prove that $L\{dw'\} \in DUPL(l)$. □

Corollary 2. *Each regular language L can be written as the left derivative of a language in $DUPL(r)$ or as the right derivative of a language in $DUPL(l)$.*

Proof. With the notations in the previous proof, we have $L = \partial_w^l(L(\Delta)) = \partial_w^r(L(\Delta'))$, where Δ is the duplication grammar constructed above and Δ' is its version for the $DUPL(l)$ case. □

If we also use a projection (a morphism which erases certain symbols and leaves the other symbols unchanged), then a representation result like that in this corollary can be obtained for linear languages, too.

Theorem 3. *For each linear language L, there is a string w, a language $L' \in DUPL(r)$, and a language $L'' \in DUPL(l)$ such that $L = h(\partial_w^l(L')) = h(\partial_w^r(L''))$.*

Proof. Consider a linear grammar, $G = (N, T, S, P)$ and two new symbols, c, d. Each rule $X \to x$ in P is replaced by $X \to cxc$ (in this way, all terminal strings appearing in the right hand side of rules are non-empty and each right-hand member of a rule is bounded by c). Denote by P' the set of rules obtained in this way. For each rule $X \to uYv \in P'$ we consider the string $uYYv$. Let w' be the string obtained by concatenating these strings, for all rules in P', then concatenating also the strings appearing in the right hand side of terminal rules in P'. Moreover, each string is considered only once, even if two rules have the same right hand side. Then, the string w we look for is $w = w'd$.

We
now construct the duplication grammar $\Delta = (N \cup T \cup \{c, d\}, \emptyset, D_r, \emptyset, \{wSS\})$, with

$$D_r = \{(X, uYYv, X) \mid X \to uYv \in P', \text{ nonterminal rule}\}$$
$$\cup \{(X, x, X) \mid X \to x \in P', \text{ terminal rule}\}.$$

From the previous construction, one can easily see that $L(\Delta)$ consists of strings of the form

$$w'dSu_1X_1u_2 \dots X_{n-1}u_nX_nxX_nv_nX_{n-1} \dots v_2X_1v_1S,$$

with $n \geq 1, X_i \in N, u_i \in \{c\}T^*, v_i \in T^*\{c\}$, for all i, and $x \in \{c\}T^*\{c\}$, such that $S \to u_1X_1v_1, X_i \to u_{i+1}X_{i+1}v_{i+1}$, for $1 \leq i \leq n-1$, and $X_n \to x$ are rules in P'. (For each derivation step we can find the string to be inserted as a substring of w. Moreover, no duplication rule can be applied for inserting a string in w, because of the presence of symbols c bounding the substrings to be duplicated and because of the fact that such substrings appear only once in w.) Therefore, the string $u_1u_2 \dots u_nxv_n \dots v_2v_1$ can be generated by using the rules in P'.

With the projection morphism $h : (N \cup T \cup \{c, d\})^* \longrightarrow T^*$ defind by $h(a) = a$ for $a \in T$, and $h(b) = \lambda$ for $a \notin T$, we obtain $L = h(\partial_w^l(L(\Delta))$.

The case of $DUPL(l)$ can be treated in the same way. □

5 Decidability and Complexity

The fact that the family CF is incomparable with all families $DUPL(\alpha)$ makes interesting several decidability questions. We solve here only one of them, the others remain open. Two examples: Is the regularity or the context-freeness of duplication languages decidable ? Given a regular language, can we decide whether or not it is in a family $DUPL(\alpha)$?

Theorem 4. *It is not decidable whether or not a context-free language is in a family $DUPL(\alpha)$, for any $\alpha \in \{l, r, 0, lr, l0, r0, lr0\}$.*

Proof. Let G be an arbitrary context-free grammar with the terminal alphabet $\{a, b\}$ and construct the language

$$L = L(G)\{c, d\}^* \cup \{a, b\}^*\{c^nd^n \mid n \geq 1\}.$$

If $L(G) = \{a,b\}^*$, then $L = \{a,b\}^*\{c,d\}^*$, therefore $L \in DUPL(\alpha), \alpha \in \{l,r,0\}$ (this can be easily seen for $\alpha \in \{l,r\}$ – see also the proof of Lemma 1 – and can be proved for $\alpha = 0$ by following the idea used in Remark 1).

If $L(G) \neq \{a,b\}^*$, then take a string $w \in \{a,b\}^* - L(G)$ and consider all strings of the form $wc^nd^n, n \geq 1$. They are in $\{a,b\}^*\{c^id^i \mid i \geq 1\}$, hence in L. Assume that, in these circumstances, $L = L(\Delta)$, for some duplication grammar $\Delta = (\{a,b,c,d\}, D_l, D_r, D_0, A)$. Consider a derivation step $z_1z_2 \Longrightarrow wc^nd^n$, where $z_1 \in \{a,b\}^*, z_2 \in \{c,d\}^*$, and the applied rule introduces a string in z_2. That is, $z_1 = w$ and $z_2 \Longrightarrow c^nd^n$. There are such derivation steps, with $z_2 \neq c^nd^n$, because A is finite and the set of strings as above is infinite. However, no string z_2 with this property can exist: we must have $z_2 = c^md^p$ with one of m,p equal to n and the other one strictly smaller than n, and such a string wz_2 is not in L (on the one hand, $w \notin L(G)$, on the other hand, $c^md^p \notin \{c^id^i \mid i \geq 1\}$).

It follows that $L \notin DUPL(\alpha)$, for all values of α.

Consequently, $L \in DUPL(\alpha), \alpha \in \{l,r,0,lr,l0,r0,lr0\}$, if and only if $L(G) = \{a,b\}^*$, which is undecidable. □

The complexity of a duplication grammar can be estimated from several points of view. We consider here some of them.

For a duplication grammar $\Delta = (V, D_l, D_r, D_0, A)$ we denote

$$
\begin{aligned}
&ax(\Delta) = card(A),\\
&axm(\Delta) = \max\{|x| \mid x \in A\},\\
&rul(\Delta) = card(D_l) + card(D_r) + card(D_0),\\
&ins(\Delta) = \max\{|v| \mid (u_1,v,u_2) \in D_l \cup D_r \cup D_0\},\\
&rad(\Delta) = \max\{|u| \mid (u_1,v,u_2) \in D_l \cup D_r \cup D_0, u = u_1 \text{ or } u = u_2\}.
\end{aligned}
$$

(These parameters count the number of axioms, the maximum length of an axiom, the number of rules, the maximum length of a string to be inserted, the maximum length of a context string – the *radius* –, respectively.)

For a language L in a family $DUPL(\alpha)$ and a measure $M \in \{ax, axm, rul, ins, rad\}$ we define

$$M_\alpha(L) = \min\{M(\Delta) \mid L = L(\Delta), \Delta \text{ is of type } \alpha\},$$

where $\alpha \in \{l,r,0,lr,l0,r0,lr0\}$.

As it is expected (as it is customary in the descriptional complexity area, see [8]), each of these parameters defines an infinite hierarchy of languages, for each α.

Theorem 5. *For each measure $M \in \{ax, axm, rul, ins, rad\}$, for each $n \geq 0$, and for each α, there is a language L_n such that $M_\alpha(L_n) = n$.*

Proof. For $n \geq 0$, consider the languages

$$L_{ax} = L_{axm} = \{a, a^2, \ldots, a^n\},$$

$$L_{rul} = \bigcup_{i=1}^{n} (ab^ia)(ab^ia)^+,$$
$$L_{ins} = a^n(a^n)^+,$$
$$L_{rad} = \{a^{2n+1}\} \cup \{a^{2(n+1)+2i} \mid i \geq 0\}.$$

Clearly, no rule can be applied to a string in the first language (otherwise, an infinite language is produced), hence each string must be introduced as an axiom. Thus, we need n axioms, the longest one being a^n.

In order to generate the second language, we need rules containing as strings to be inserted strings of the form $(ab^ia)^j$ with $j \geq 1$, for each $i = 1, 2, \ldots, n$ (we cannot modify a block b^i in a string $(ab^ia)(ab^ia)^r, r \geq 1$, hence we can only introduce new blocks ab^ia). That is, we need at least n duplication rules.

In the case of the third language it is also clear that the inserted strings must be of the form $a^{nk}, k \geq 1$, that is, at least n symbols must be simultaneously inserted.

Grammars with n rules and with a string of length n to be inserted, respectively, can generate these languages.

For the fourth language, starting from the axioms

$$A = \{a^{2n+1}, a^{2(n+1)}, a^{2(n+1)+2}\},$$

and using the rule (a^{n+1}, a^2, a^{n+1}) in any mode l, r, or 0, we can generate this language. However, we cannot generate this language by using rules (a^m, a^p, a^q) with $m, q \leq n$: any such a rule must have p even, $p = 2k, k \geq 1$; applying it to the string a^{2n+1} we produce the string $a^{2n+1+2k}$, which is not in L_{rad}, a contradiction.

These remarks are valid for all variants of duplication grammars. Consequently, for each M we have $M_\alpha(L_M) = n$. □

A natural problem concerns the closure properties of families $DUPL(\alpha)$. We do not consider it here, but we only point out that the families $DUPL(l), DUPL(r)$ are not closed under mirror image – a consequence of Lemma 2.

6 Final Remarks

We have considered here duplication operations suggested by similar operations met in linguistics and in the genetic area. Some other variants can be defined, for instance, with a transformation applied to the copy of the duplicated string (point mutations, reversal, etc). Also variants of applying the replication rules can be of interest: leftmost use of rules, parallel application, priority relations among rules and so on. The area deserves further investigations, expecially for those variants of the replication operation which are met in DNA transformation/evolution.

In general, the operations on strings inspired from biochemistry were successfully extended to various bi- or multi-dimensional structures, such as trees, graphs in general, arrays. (The reader is referred to [4], [6], [7] for modern accounts on these areas.) This happens, for instance, with the splicing operation ([10]) and it is probably true also for the duplication operations considered here. The case of trees

is particularly important, because the DNA molecules are known to often take a branching structure.

References

[1] J. Dassow, V. Mitrana, On some operations suggested by the genome evolution, *Proc. of Pacific Symp. on Biocomputing*, Hawaii 97 (R. Altmann, et al, eds.), World Scientific, Singapore, 1997, 97 – 108.

[2] J. Dassow, V. Mitrana, A. Salomaa, Context-free evolutionary grammars and the structural language of nucleic acids, *Bio Systems*, 43 (1997), 169 – 177.

[3] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, Heidelberg, 1989.

[4] J. Engelfriet, Context-free graph grammars, chapter 3 in vol. 3 of [16], 125 – 213.

[5] B. S. Galiukschov, Semicontextual grammars (in Russian), *Mat. logica i mat. ling.*, Talinin Univ., 1981, 38 – 50.

[6] F. Gécseg, M. Steinby, Tree languages, chapter 1 in vol. 3 of [16], 1 – 68.

[7] D. Giammarresi, A. Restivo, Two-dimensional languages, chapter 4 in vol. 3 of [16], 215 – 267.

[8] J. Gruska, Descriptional complexity of context-free languages, *Proc. Math. Found. Computer Sci. Conf.*, High Tatras, 1973, 71 – 83.

[9] D. Hauschild, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Inform.*, 31 (1994), 719 – 728.

[10] T. Head, Gh. Păun, D. Pixton, Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, chapter 7 in vol. 2 of [16], 295 – 360.

[11] L. Hunter, Molecular biology for computer scientists, in *Artificial Intelligence and Molecular Biology* (L. Hunter, ed.), AAAI Press/The MIT Press, Menlo Park, CA, 1993, 1 – 46.

[12] A. Manaster Ramer, *Uses and misuses of mathematics in linguistics*, X Congreso de Lenguajes Naturales y Lenguajes Formales, Sevilla, 1994.

[13] Gh. Păun, *Marcus Contextual Grammars*, Kluwer Academic Publ., Boston, Dordrecht, 1997.

[14] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.

[15] W. C. Rounds, A. Manaster Ramer, J. Friedman, Finding natural languages a home in formal language theory, in *Mathematics of language* (A. Manaster Ramer, ed.), John Benjamins, Amsterdam, 1987, 349 – 360.

[16] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, Heidelberg, 1997.

[17] A. Salomaa, *Jewels of Formal Languages*, Computer Science Press, Rockville, 1981.

[18] D. B. Searls, The linguistics of DNA, *American Scientist*, 80 (1992), 579 – 591.

[19] A. Thue, Uber unendliche Zeitchenreihen, *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiania*, 7 (1906), 1 – 22.

Acta Cybernetica **14** (1999) 165-177.

Some Properties of Duplication Grammars

Victor MITRANA * Grzegorz ROZENBERG †

Abstract

This paper considers context-free variants of duplication grammars. We investigate their generative capacity, their mutual relationship, and their relationship to the context-sensitive duplication grammars. We solve some problems left open in [6], e.g., proving that all regular languages can be generated by nearly all types of context-sensitive duplication grammars. We also consider some decision problems.

1 Introduction

String duplications or duplications of segments of strings are rather frequent in both natural and genetic languages. We refer to [1], [2] and [10] for discussions of duplication, and other operations related to the language of nucleic acids. For motivations coming from linguistics, we refer to [5] and [9].

Based on [1], Martin-Vidè and Păun introduced in [6] a generative mechanism (similar to the one considered in [2]) based only on duplication: one starts with a given finite set of strings and produces new strings by copying specified substrings to certain places in a string, according to a finite set of duplication rules. This mechanism is studied in [6] from the generative power point of view. The present paper considers the context-free versions of duplication grammars - this formalizes a possible hypothesis that duplications appear more or less at random within the genome in the course of its evolution. We solve some problems left open in [6], prove new results concerning the generative power of context-sensitive and context-free duplication grammars, and compare the two classes of grammars. Finally, some decision problems are discussed.

A *context-sensitive duplication rule* is a triple whose components are strings over a given alphabet (in the case of DNA the alphabet consists of the four nucleotids), say (u, x, v), which has the following interpretation:

- the string x, which appears to the left of uv in the processed string, is inserted in between u and v;

*University of Bucharest, Faculty of Mathematics Str. Academiei 14, 70109, Bucharest, Romania mitrana@funinf.math.unibuc.ro

†Leiden Institute of Advanced Computer Science, Leiden University, PO Box 9512, 2300 RA Leiden, The Netherlands, rozenber@wi.leidenuniv.nl and Department of Computer Science, University of Colorado at Boulder, USA.

- the string x, which appears to the right of uv in the processed string, is inserted in between u and v;

- the string x which appears in between u and v is doubled.

A *context-free duplication rule* is a string over the given alphabet, say x, whose effect is the duplication of x either to the right of, or to the left of, or immediately after, an already existing copy of x. Clearly, context-free duplication rules may be viewed as context sensitive duplication rules whose contexts are empty.

In vivo, cross-over takes place just between homologous chromosomes (chromosomes of the same type and of the same length), see [4]. A model of a cross-over between a DNA molecule and its replicated version is considered in [3] - this is a model for a cross-over between "sister" chromatides. One specifies an initial finite set of strings and a finite set of cross-over rules of the form $(\alpha, \beta, \gamma, \delta)$. It is assumed that every initial string is replicated so that two identical copies of every initial string are available. The first copy is cut between the segments α and β and the other one is cut between γ and δ. Now, the last segment of the second string gets attached to the first segment of the first string, and a new string is obtained. More generally, another string is also generated, by linking the first segment of the second string with the last segment of the first string. Iterating the procedure, one gets a language. The main idea of this approach is schematically presented in the Figure 1.

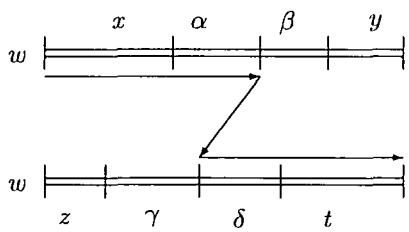


Figure 1

Hence, the splicing operation introduced by T. Head, see, e.g., [7] is performed here between identical strings. It is easily seen that one obtains the insertion of a substring of w in w; this induces a duplication of some chromosomes into genome. This type of recombination is considered to be the main way of producing tandem repeats or block deletions in chromosomes.

2 Basic definitions

In this section we give the basic notions and notations needed in the sequel. For basic formal language theory we refer to [7] or [8]. We use the following basic notation. The length of a word x is denoted by $|x|$, the empty string is denoted by ε; we have $|\varepsilon| = 0$. The mirror image of a word x is denoted by x^R. The set of all words over V is denoted by V^*, and $V^+ = V^* \setminus \{\varepsilon\}$. For sets X and Y, $X \setminus Y$ denotes the set-theoretic difference of X and Y. If X is finite, then $card(X)$ denotes its cardinality; $\emptyset$ denotes the empty set.

Here is the main notion of this paper.

A *(context-sensitive) duplication grammar* is a construct

$$\Delta = (V, D_l, D_r, D_0, A),$$

where V is an alphabet, D_l, D_r, D_0 are finite subsets of $V^* \times V^+ \times V^*$, and A is a finite subset of V^+. The elements of D_l, D_r and D_0 are context-sensitive duplication rules, and elements of A are called axioms.

Given a duplication grammar as above and two words $x, y \in V^+$, we define the following three types of direct derivation relations in Δ:

$$\begin{aligned}
x \ &\Longrightarrow_{D_l} \ y \text{ iff } x = x_1uvx_2zx_3,\ y = x_1uzvx_2zx_3, \\
&\qquad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } (u, z, v) \in D_l, \\
x \ &\Longrightarrow_{D_r} \ y \text{ iff } x = x_1zx_2uvx_3,\ y = x_1zx_2uzvx_3, \\
&\qquad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } (u, z, v) \in D_r, \\
x \ &\Longrightarrow_{D_0} \ y \text{ iff } x = x_1uzvx_2,\ y = x_1uzzvx_2, \\
&\qquad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } (u, z, v) \in D_0.
\end{aligned}$$

The union of these relations is the direct derivation relation of Δ, denoted by $\Longrightarrow$, and the reflexive and transitive closure of $\Longrightarrow$ is the derivation relation of Δ, denoted by $\Longrightarrow^*$. The language generated by the duplication grammar Δ is defined by

$$L(\Delta) = \{y \in V^* \mid x \Longrightarrow^* y, \text{ for some } x \in A\}.$$

Thus, the language of Δ consists of all words obtained by beginning with strings in A, and applying iteratively duplication rules from $D_l \cup D_r \cup D_0$. The application of a rule to a string means to copy one of its substrings to the left of, or to the right of, or next to its "given" occurrence. Because each of the three sets of rules may be empty, one obtains seven families of languages denoted by $DUPL(X)$, $X \in \{l, r, 0, lr, l0, r0, lr0\}$; the presence of a letter within X means that the corresponding set of rules is non-empty, e.g., for $X = l0$, $D_l \neq \emptyset$, $D_0 \neq \emptyset$ and $D_r = \emptyset$.

Analogously, we define a context-free duplication grammar as a construct $\Delta = (V, D_l, D_r, D_0, A)$, where V and A have the same interpretation as above, but D_l, D_r, D_0 are finite subsets of V^+ whose elements are context-free duplication rules. Given a context-free duplication grammar as above and two words $x, y \in V^+$,

we define three types of direct derivation relations:

$x \models_{D_l} y$ iff $x = x_1x_2zx_3$, $y = x_1zx_2zx_3$, with $x_1, x_2, x_3 \in V^*$, and $z \in D_l$,
$x \models_{D_r} y$ iff $x = x_1zx_2x_3$, $y = x_1zx_2zx_3$, with $x_1, x_2, x_3 \in V^*$, and $z \in D_r$,
$x \models_{D_0} y$ iff $x = x_1zx_2$, $y = x_1zzx_2$, with $x_1, x_2, x_3 \in V^*$, and $z \in D_0$.

Again, the union of these relations is the direct derivation relation, denoted by $\models$, and the reflexive and transitive closure of $\models$ is the derivation relation, denoted by $\models^*$. The language generated by the context-free duplication grammar Δ is defined by

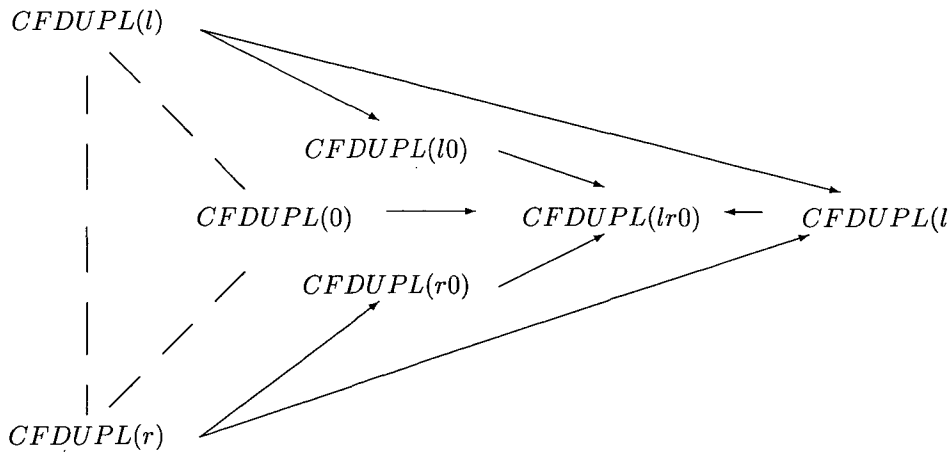
$$L(\Delta) = \{y \in V^* \mid x \models^* y, \text{ for some } x \in A\}.$$

Again, we get seven families of languages denoted by $CFDUPL(X)$, $X \in \{l, r, 0, lr, l0, r0, lr0\}$.

3 A short comparison

We begin by settling the relationships among the seven families of context-free duplication languages.

Theorem 1. *The relations in the following diagram hold, where an arrow indicates a strict inclusion and a dotted line links two incomparable families.*



Proof. The language $\{a^nb^ma^pb^q | n, m, p, q \geq 1\}$ is in $CFDUPL(0)$ (one starts with *abab* and doubles either an occurrence of a or an occurrence of b) but not in $CFDUPL(lr)$. To see the latter, we note that each context-free duplication grammar having just left and right duplication rules generates strings in $a^+b^+a^+b^+a^+b^+$; a contradiction.

By a similar reasoning, the language $\{a^nb^m | n, m \geq 1\}$ belongs to $CFDUPL(l0) \cap CFDUPL(r0) \cap CFDUPL(lr)$ but not to $CFDUPL(l) \cup CFDUPL(r)$.

The language $\{a,b,c\}^+$ is in $CFDUPL(r) \cap CFDUPL(l)$ (the initial set contains all strings of length at most 3, each letter a,b,c appearing at most once; duplication rules allow copying of any letter to the right/left of one of its occurrences.) Because there are arbitrarily long square-free strings in $\{a,b,c\}^+$, [11], it follows that $\{a,b,c\}^+ \notin CFDUPL(0)$.

Finally,

$$\{a,b,c\}^+\{\$\}^+\{d,e,f\}^+ \in CFDUPL(lr0) \setminus (CFDUPL(l0) \cup CFDUPL(r0))$$

which concludes the proof. □

The following result concerning the relationships among the context-sensitive families of duplication languages has been proved in [6].

Theorem 2.[6]

1. *The families DUPL(l) and DUPL(r) are incomparable.*
2. *The following inclusions*

$$DUPL(r) \cup DUPL(l) \subset DUPL(lr)$$
$$DUPL(0) \subset (DUPL(r0) \cap DUPL(l0))$$

are proper.

It is an *open problem* whether or not $DUPL(0)$ is included in $DUPL(l)$ or in $DUPL(r)$. However, we have

Proposition 1. *CFDUPL(0) is strictly included in DUPL(lr).*

Proof. Let $\Delta = (V,\emptyset,\emptyset,D_0,A)$ be a duplication grammar with $D_0 = \{x_1,x_2,\ldots,x_n\}$. Construct a duplication grammar $\Delta' = (V,D_l,D_r,\emptyset,A')$, where

$$\begin{aligned} D_l = D_l &= \{(x_i,x_i,\varepsilon) | 1 \leq i \leq n\}, \\ A' &= \{z \in L(\Delta) | \text{ each } x_i \text{ has at most two non-overlapped occurrences in } z\}. \end{aligned}$$

It is easy to see that A' is a finite set, and $L(\Delta) = L(\Delta')$. □

Along the same lines, we have

Theorem 3. $CFDUPL(X) \subset DUPL(X)$, *for all* $X \in \{0,l,r,l0,r0,lr,lr0\}$.

Proof. It suffices to provide languages that prove all inclusions to be strict.

The duplication grammar $\Delta = (\{a,b\},\{(\varepsilon,a,a),(\varepsilon,b,b)\},\emptyset,\emptyset,\{ab,a^2b,ab^2,a^2b^2\}$ generates $L_1 = \{a^nb^m | n,m \geq 1\}$. Hence L_1 is in $DUPL(l)$ (also in $DUPL(r)$) but not in $(CFDUPL(l) \cup CFDUPL(r)$.

Similarly, $\{a^nb^ma^pb^q | n,m,p,q \geq 1\} \in DUPL(lr) \setminus CFDUPL(lr)$.

One can show that $\{a^nb^nab | n \geq 1\}$ cannot be generated by any context-free duplication grammar. On the other hand, $\{a^nb^nab | n \geq 1\} \in DUPL(lr0)$ (see [6]).

Take now the language $L_2 = \{ab^n c^m d^p e | 1 \leq n, m, p \leq 3\}^+$. This language can be obtained by starting with the string *abcde* and iteratively applying rules from the set

$$D_0 = \{(\varepsilon, abcde, \varepsilon), (a, b, c), (ab, b, c), (b, c, d), (bc, c, d), (c, d, e), (cd, d, e)\}.$$

Consider the homomorphism $h : \{a, b, c\}^* \longrightarrow \{a, b, c, d, e\}^*$ defined by $h(a) = ab^3cde$, $h(b) = abc^3de$, $h(c) = abcd^3e$. Let x be an arbitrarily long square-free string over $\{a, b, c\}$. The string $h(x)$ is in L_2. It is easy to notice that the adjacent identical substrings in $h(x)$ are only the letters from $\{a, b, c\}$. If L_2 were in $CFDUPL(0)$, then any context-free duplication grammar generating L_2 would generate strings containing arbitrarily many adjacent occurrences of the same letter from $\{a, b, c\}$; a contradiction. □

4 Observations on the generative power

We start by considering unary alphabets. We will prove that in this case the generative power of duplication grammars equals the accepting power of deterministic finite automata. To this end, we prove the following lemma.

Lemma 1. *Over the unary alphabet, the equality* $DUPL(X) = CFDUPL(0)$ *holds for any* $X \in \{l, r, 0, lr, l0, r0, lr0\}$.

Proof. Let $\Delta = (\{a\}, D_l, D_r, D_0, A)$ be a duplication grammar. Let

$$\begin{aligned} D_l &= \{(u_l, a^{i_l}, v_l) | 1 \leq l \leq n\}, \\ D_r &= \{(x_l, a^{j_l}, y_l) | 1 \leq l \leq m\}, \\ D_0 &= \{(z_l, a^{k_l}, w_l) | 1 \leq l \leq p\}. \end{aligned}$$

Take

$$\alpha = max(\{|uxv| \ : \ (u, x, v) \in D_l \cup D_r \cup D_0\} \cup \{|x| \ : \ x \in A\}.$$

Consider now the context-free duplication grammar

$$\Delta' = (\{a\}, \emptyset, \emptyset, D_0', A'),$$

where

$$\begin{aligned} A' &= \{x | x \in L(\Delta), |x| \leq 3\alpha\}, \\ D_0' &= \{a^q | q = \sum_{s=1}^{n} \alpha_s i_s + \sum_{s=1}^{m} \beta_s j_s + \sum_{s=1}^{p} \gamma_s k_s, \ \alpha \leq q \leq 2\alpha\}. \end{aligned}$$

We claim that $L(\Delta) = L(\Delta')$. Note that each rule in D_0' is applicable to strings of length at least α. Furthermore, each application of a rule in D_0' simulates the application of a sequence of rules from $D_l \cup D_r \cup D_0$. Consequently, $L(\Delta') \subseteq L(\Delta)$.

All strings of length at most 3α from $L(\Delta)$ are also in $L(\Delta')$. Let z be the shortest string in $L(\Delta)$ such that $|z| > 3\alpha$. Then there exists a derivation in Δ':

$$x \Longrightarrow^{+} y \Longrightarrow^{+} z$$

with

$$\begin{aligned} &(i) && x \in A, \\ &(ii) && \alpha \leq |y| \leq 3\alpha, \\ &(iii) && \alpha \leq |z| - |y| \leq 2\alpha. \end{aligned}$$

Because $y \in A'$ one may write $y \Longrightarrow_{D'_0} z$, and so $z \in L(\Delta')$. Inductively, $L(\Delta) \subseteq L(\Delta')$. □

Theorem 4. *A language over a unary alphabet is regular if and only if it is generated by a duplication grammar.*

Proof. By the previous lemma, it suffices to consider duplication grammars with just context-free duplication rules whose effect is to double an occurrence of a substring. Let $L \subseteq \{a\}^*$ be a regular language. Then, there exist a finite set F and the positive integers $k_i, 1 \leq i \leq m$, and $q > max\{\#(x) | x \in F\}$ such that

$$L = F \cup \bigcup_{i=1}^{m} \{a^{k_i+nq} | n \geq 1\}$$

This can be easily seen if one considers a deterministic finite automaton accepting L, for which the transition function is defined everywhere.

Consider now the duplication grammar:

$$\Delta = (\{a\}, \emptyset, \emptyset, \{a^q\}, F \cup \{a^{k_i+q} | 1 \leq i \leq m\}.$$

Clearly, $L = L(\Delta)$. Duplications can never be carried out on words of F.

Conversely, let us consider a duplication grammar $\Delta = (\{a\}, \emptyset, \emptyset, D_0, A)$, with $D_0 = \{a^{c_1}, a^{c_2}, \ldots, a^{c_n}\}$. Let

$$p = gcd(d_1, d_2, \ldots, d_m, c_1, c_2, \ldots c_n),$$

where *gcd* means the greatest common divisor. If $L(\Delta)$ is finite, then it is obviously regular. If $L(\Delta)$ is an infinite set, then there are $t_i, 1 \leq i \leq s,\ s \leq p$, such that

$$L(\Delta) = F \cup \bigcup_{i=1}^{s} \{a^{t_i+kp} | k \geq 0\},$$

for some finite set F. Consequently, $L(\Delta)$ is regular which completes the proof. □

The next result settles a problem left open in [6].

Theorem 5. *All regular languages are in* $DUPL(X)$, $X \in \{l, r, l0, r0, lr, lr0\}$.

Proof. We present a proof for $DUPL(r)$, the proofs for other cases are analogous. Let R be a regular language recognized by the deterministic finite automaton $M = (Q, V, \delta, q_0, F)$ with the total transition function δ. Let for each state q, C_q be defined as follows:

$$C_q = \{x \in V^+ | \delta(q, x) = q \text{ by passing each state, different from } q, \text{ at most once}\}.$$

For strings $x, y \in V^*$, we define the equivalence relation $\sim_R$ as follows:

$$(x \sim_R y) \text{ iff } (uxv \in R \text{ iff } uyv \in R), \text{ for any } u, v \in V^*.$$

It is well-known (see e.g. [8]) that $V^*/\sim_R$ (the quotient of V^* by $\sim_R$) is finite; let k be the cardinality of $V^*/\sim_R$ (the index of $\sim_R$).

Now, one constructs the duplication grammar $\Delta = (V, \emptyset, D_r, \emptyset, A)$, where

$$\begin{aligned} D_r &= \bigcup_{q \in Q, C_q \neq \emptyset} \{(x, y, \varepsilon) | xy \sim_R x,\ |x| < k, y \in C_q\}, \text{ and} \\ A &= \{w \in R | \text{ for each } q \in Q, \text{ each string in } C_q \\ &\qquad \text{has at most } k \text{ non-overlapping occurrences in } w\}. \end{aligned}$$

We claim that A is finite. Indeed, no word longer than $(k+1)l \cdot card(Q)$, where $l = max\{card(C_q) | q \in Q\}$, is in A. To see this, assume that such a word, say w, is in A; so $|w| = p \geq (k+1)l \cdot card(Q)$. Let $q_0, q_1, \ldots, q_p$, $q_p \in F$, be the sequence of states that accepts w. At least $(k+1)l$ states in this sequence must be the same; assume that q is such a state. But then w contains at least $k+1$ identical substrings in C_q; a contradiction.

Clearly, $L(\Delta) \subseteq R$. Let z be the shortest word in $R \setminus L(\Delta)$. Thus, there exists $x \in C_q$, for some $q \in Q$, such that x occurs more than k times in z. Let $z = wxy$, with $|w| \geq k$, where the given occurrence of x is the last (rightmost) occurrence of x in z. Let $z = uvxy$ with $|v| = k$. Thus v has $k+1$ prefixes, and so there are two prefixes v_1, v_2 of v such that $v_1 \sim_R v_2$ and $|v_1| < |v_2|$. We choose the closest pair of such prefixes. By replacing v_2 by v_1 in v we get a string $uv'xy$ which is in $L(\Delta)$ because it is in R and it is shorter than z. Moreover, $v_2 = v_1 t$, where t must be in C_q, for some $q \in Q$ (because of the choice of v_1 and v_2). Consequently, $(v_1, t, \varepsilon) \in D_r$, and so $uv'xy \Longrightarrow_{D_r} z$. Thus $z \in L(\Delta)$; a contradiction.

Analogously one proves that each regular language is in $DUPL(l)$. □

We recall that the family $DUPL(0)$ is incomparable with the family of regular languages.

The position of the class of regular languages with respect to the classes of context-free duplication languages is given by the next theorem.

Theorem 6. *The family of regular languages is incomparable with any of the families* $CFDUPL(X), X \neq 0$.

Proof. The regular language $V^+\{c\}^+V^+$, where V contains at least three symbols and $c \notin V$, cannot be generated by any context-free duplication grammar. Indeed,

if a context-free duplication grammar generates all strings in $V^+\{c\}^+V^+$, then it must contain left/right duplication rules involving strings in $V^+ + c^+$. Therefore, also strings in $V^+\{c\}^+V^+\{c\}^+V^+$ can be generated.

Consider now the Dyck language over $\{a,b\}$, denoted by D_{ab}, and the non-regular language $L = \{ab\}D_{ab}$. This language is in $CFDUPL(r)$. The context-free duplication grammar $\Delta = (\{a,b\}, \emptyset, \{ab\}, \emptyset, \{abab\})$ with only right duplication rules generates L. Clearly, $L(\Delta) \subseteq L$; let z be the shortest string in $L \setminus L(\Delta)$. If $z = abxy$, with $x, y \in D_{ab}$, then ab yields z in Δ as follows:

$$ab \models^* aby \models^* abxy.$$

If $z = abaxb$, with $x \in D_{ab}$, then the derivation $ab \models abab \models^* abaxb$ is possible in Δ. Consequently, $L(\Delta) = L$. □

The relation between $CFDUPL(0)$ and the class of regular languages remains open.

Recall that a homomorphism which erases some symbols and leaves the others symbols unchanged is called a *projection*. A projection $h : (V \cup V')^* \longrightarrow V^*$ that erases the symbols in V' only is the projection of V, denoted by pr_V.

Theorem 7. *For each context-free language $L \in V^*$, there exists a language L in CFDUPL(r) (CFDUPL(l)) and a homomorphism h such that $L = pr_V(h^{-1}(L'))$.*

Proof. Let $G = (N, V, S, P)$ be a context-free grammar generating L. Assume that

$$P = \bigcup_{i=1}^{n} \{A_i \longrightarrow x_{i,j} | 1 \leq j \leq r_i\},$$

with $S = A_1$. Furthermore, we assume that $\varepsilon \notin L$. Let $V' = N \cup V \cup \{c_i | 1 \leq i \leq n\} \cup \{d\}$, where c_i, d, are new symbols. Let then Δ be the duplication grammar $(V', \emptyset, D_r, \emptyset, A)$, where

$$\begin{aligned} D_r &= \{(c_i x_{i,j} | 1 \leq i \leq n,\ 1 \leq j \leq r_i\}, \text{ and} \\ A &= \{cx_{1,1}dcx_{1,2}d \ldots dcx_{1,r_1}dcx_{2,1}d \ldots dcx_{n,r_n}dA_1\}. \end{aligned}$$

Now, let h be the homomorphism

$$h : (V \cup \{[i,j] | 1 \leq i \leq n, 1 \leq j \leq r_i\} \cup \{c_i | 1 \leq i \leq n\} \longrightarrow (V')^*$$

such that

$$\begin{aligned} h([i,j]) &= c_i x_{i,j} d, 1 \leq i \leq n,\ 1 \leq j \leq r_i, \\ h(c_i) &= A_i c_i, 1 \leq i \leq n, \text{ and} \\ h(a) &= a,\ a \in V. \end{aligned}$$

It is easy to see that $pr_V(h^{-1}(L(\Delta))) = L(G)$. Clearly, whenever a substring $c_i x_{i,j}$ is copied, this is done somewhere to the right of the last occurrence of d -

otherwise one gets a string "rejected" by applying the inverse homomorphism h. Also, all strings that contain nonterminal occurrences that are not immediately followed by some c_i, to the right of the last occurrence of d, are rejected in the same way. Moreover, every occurrence of a nonterminal A_i, situated to the right of the last occurrence of d, has to be followed by just one occurrence of c_i. In this way duplication rules simulate the application of production rules in G. □

5 Decision problems

We discuss in this section some basic decision problems. We begin by pointing out that the "totality problem" is decidable for all families of duplication languages.

Theorem 8. *Let Δ be a duplication grammar over the alphabet V. It is decidable whether or not $L(\Delta) = V^*$.*

Proof. We will consider duplication grammars having only left duplication rules - the other types of duplication grammars can be treated in a similar way. Let $\Delta = (V, D_l, \emptyset, \emptyset, A)$ be a duplication grammar. The main point of our argument is the following property

$$L(\Delta) = V^* \text{ if and only if } \{x \in V^* : |x| \leq k+1\} \subset L(\Delta),$$

where $k = max\{|x| : x \in A\}$.

The "only if" part is obvious. For the "if" part of the proof, assume that z is a shortest word in $V^* \setminus L(\Delta)$. This word can be written as $z = ya$ with $a \in V$. Hence $y \in L(\Delta) \setminus A$, and so there exists $x \in A$ such that $x \Longrightarrow_{D_r}^{+} y$. Because $|xa| < |ya|$, it follows that $xa \in L(\Delta)$. But, also $xa \Longrightarrow_{D_r}^{+} ya = z$. To conclude, it suffices to note that the inclusion $\{x \in V^* : |x| \leq k+1\} \subset L(\Delta)$ is decidable due to the decidabilty of the membership problem. □

It is proved in [6] that the membership of a context-free language in the family of languages $DUPL(X), X \neq 0$, is not decidable. Our next theorem extends this result to the families of context-free duplication languages, as well as to $DUPL(0)$.

Theorem 9. *It is not decidable whether or not a context-free language is in a family $CFDUPL(X), X \notin \{r, l\}$.*

Proof. The proof is similar to the one in [6]. Let G be an arbitrary context-free grammar with the terminal alphabet $\{a, b\}$, and let

$$L = L(G)\{c, d\}^* \cup \{a, b\}^*\{c^n d^n | n \geq 1\}.$$

If $L(G) = \{a, b\}^*$, then $L = \{a, b\}^*\{c, d\}^*$ which is in $CFDUPL(X)$, for all $X \notin \{r, l\}$. It is easily seen that the grammar $\Delta = (\{a, b, c, d\}, \emptyset, \emptyset, D_0, A)$, with

$$D_0 = \{a, b, c, d, ab, ba, cd, dc\}, \text{ and } A = \{a, b, c, d, ab, aba, ba, bab, cd, cdc, dc, dcd\},$$

generates $\{a,b\}^*\{c,d\}^*$. The reader my easily check this assertion.

If $L(G) \neq \{a,b\}^*$, then L cannot be generated by any context sensitive duplication grammar (see the proof of Theorem 4 in [6]). Consequently, $L \in CFDUPL(X)$ for $X \notin \{r,l\}$, if and only if $L(G) = \{a,b\}^*$, which is undecidable. □

This result can be also extended to the families $CFDUPL(r)$ and $CFDUPL(l)$.

Theorem 10. *It is not decidable whether or not a context-free language is in a family* $CFDUPL(X), X \in \{r,l\}$.

Proof. The proof is based on a reduction to the Post Correspondence Problem (PCP). Take an arbitrary instance of PCP, i.e., two arbitrary n-tuples of nonempty strings over the alphabet $\{a,b\}$:

$$x = (x_1, x_2, \ldots, x_n),$$

$$y = (y_1, y_2, \ldots, y_n).$$

Then, consider the languages

$$L_z = \{ba^{i_1}ba^{i_2}\ldots ba^{i_k}cz_{i_k}\ldots z_{i_2}z_{i_1} | k \geq 1\} \text{ for } z \in \{x,y\},$$

$$L_s = \{w_1 cw_2 cw_2^R cw_1^R | w_1, w_2 \in \{a,b\}^*\}, \text{ and}$$

$$L(x,y) = \{a,b,c\}^* - (L_x\{c\}L_y^R \cap L_s).$$

It is known that $L(x,y)$ is a context-free language. For every solution $(i_1, i_2, \ldots, i_k)$ of $PCP(x,y)$ the strings

$$ba^{i_1}ba^{i_2}\ldots ba^{i_k}cx_{i_k}\ldots x_{i_2}x_{i_1}cy_{i_1}^R y_{i_2}^R \ldots y_{i_k}^R ca^{i_k}b\ldots ba^{i_2}ba^{i_1}b$$

are not in $L(x,y)$.

Clearly, when $L(x,y) = \{a,b,c\}^*$, then $L(x,y)$ is in $CFDUPL(r) \cap CFDUPL(l)$.

Now, it is sufficient to prove that $L(x,y) \notin CFDUPL(l) \cup CFDUPL(r)$ if $L(x,y) \neq \{a,b,c\}^*$.

Let us suppose that $L(x,y) = L(\Delta)$, $\Delta = (\{a,b,c\}, \emptyset, D_r, \emptyset, A)$. We choose a solution $(i_1, i_2, \ldots i_k)$ such that

$$|x_{i_k}x_{i_{k-1}}\ldots x_{i_1}| > max\{|w| \; | w \in A\}.$$

For $\{a,b\}^* \subseteq L(\Delta)$, there exists a word $w \in A$ such that

$$w \models^* y_{i_1}^R y_{i_2}^R \ldots y_{i_k}^R \in L(\Delta).$$

By the choice of the solution $(i_1, i_2, \ldots, i_k)$ the word

$$z = ba^{i_1}ba^{i_2}\ldots ba^{i_k}cx_{i_k}\ldots x_{i_2}x_{i_1}cwca^{i_k}b\ldots ba^{i_2}ba^{i_1}b$$

is in $L(\Delta)$.

Therefore, we get

$$z \models^* ba^{i_1}ba^{i_2}\dots ba^{i_k}cx_{i_k}\dots x_{i_2}x_{i_1}cy_{i_1}^R y_{i_2}^R \dots y_{i_k}^R ca^{i_k}b\dots ba^{i_2}ba^{i_1}b,$$

a contradiction. Hence the theorem holds. □

Finally, we consider "nonemptiness of the intersection problem" for $DUPL(X), X \neq 0$.

Theorem 11. *It is undecidable whether or not $L_1 \cap L_2 = \emptyset$? for arbitrary two duplication languages in $DUPL(X), X \neq 0$.*

Proof. Let $x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n)$ be an instance of PCP, and let

$$\begin{aligned} L_x &= \{w\$cd^{i_1}\$cd^{i_2}\dots\$cd^{i_k}x_{i_k}\dots x_{i_2}x_{i_1} | k \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq k\} \\ &\cup \{w\$cd^{i_1}\$cd^{i_2}\dots\$cd^{i_k}\$x_{i_k}\dots x_{i_2}x_{i_1} | k \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq k\}, \end{aligned}$$

where $w = cdx_1cdy_1cd^2x_2cd^2y_2\dots cd^n x_n cd^n y_n$. L_y is defined analogously.

Clearly, the duplication grammar $\Delta = (\{a, b, c, d, \$, \#\}, \emptyset, D_r, \emptyset, \{w\$\#\})$, with

$$\begin{aligned} D_r &= \{(\$, cd^i x_i, \#) | 1 \leq i \leq n\} \cup \{(\$, cd^i x_i, X) | 1 \leq i \leq n, X \in \{a, b\}\} \\ &\cup \{(d, \$, a), (d, \$, b)\} \end{aligned}$$

generates L_x.

This concludes the proof, because $L_x \cap L_y = \emptyset$ if and only if the instance (x, y) of PCP has no solution. □

Acknowledgements

The first author is grateful to Leiden Center for Natural Computing for supporting his stay at Leiden University in June 1998, during which the work on this paper was initiated.

References

[1] J. Dassow and V. Mitrana, On some operations suggested by the genome evolution. *Pacific Symposium on Biocomputing'97* (R. Altman, K. Dunker, L. Hunter, T. Klein eds.), Hawaii, 1997, 97–108.

[2] J. Dassow and V. Mitrana, Evolutionary grammars: a grammatical model for genome evolution, *Proc. German Conf. in Bioinformatics GCB'96*, (R. Hofestädt, T. Lengauer, M. Löffler, D. Schomburg eds.), LNCS 1278, Springer-Verlag, 1997, 199–209.

[3] J. Dassow and V. Mitrana, Self cross-over systems. In *Computing with biomolecules* (Gh. Paun ed.), World Scientific, 1998 (in press).

[4] D. L. Hartl, D. Freifelder and L. A. Snyder, *Basic Genetics*, Jones and Bartlett Publ., Boston, Portola Valley, 1988.

[5] A. Manaster Ramer, Uses and misuses of mathematics in linguistics, *Proc. X Congreso de Lenguajes Naturales y Lenguajes Formales*, Sevilla, 1994.

[6] C. Martin-Vidè and G. Păun, Duplication grammars, *Acta Cybernetica* (submitted).

[7] G. Păun, G. Rozenberg, A. Salomaa, *DNA Computing, New Computing Paradigms*, Springer Verlag, Berlin, Heidelberg, 1998.

[8] G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, vol. I, Springer, Berlin, 1997.

[9] W. C. Rounds, A. Manaster Ramer and J. Friedman, Finding natural languages a home in formal language theory. In *Mathematics of Language* (A. Manaster Ramer ed.), John Benjamins, Amsterdam, 1987, 349–360.

[10] D. B. Searls, The computational linguistics of biological sequences. In *Artificial Intelligence and Molecular Biology* (L. Hunter ed.), AAAI Press, The MIT Press, 1993, 47–120.

[11] A. Thue, Uber unendliche Zeitchenreihen, *Norske Vid. Selk. Skr. I. Mat. Nat. Kl. Christiania*, 7(1906), 1–22.

Acta Cybernetica **14** (1999) 179-192.

Watson-Crick Walks and Roads on DOL Graphs*

Arto Salomaa †

Abstract

Apart from the massive parallelism of DNA strands, the phenomenon known as Watson-Crick complementarity is basic both in the experiments and theory of DNA computing. The parallelism makes exhaustive searches possible, whereas the complementarity is a powerful computational tool. This paper investigates complementarity as a language-theoretic operation: "bad" words obtained through a generative process are replaced by their complementary ones. While this idea is applicable to any generative process, it seems particularly suitable for Lindenmayer systems. DOL systems augmented with a specific complementarity transition, "Watson-Crick DOL systems", are investigated in this paper. Many issues involved are conveniently expressed in terms of certain paths, "Watson- Crick walks", in an associated digraph.

Keywords: DNA computing, Lindenmayer systems, DOL sequences, Watson-Crick complementarity.

1 Introduction

Adleman's celebrated experiment, [1], demonstrated how methods of molecular biology can possibly be applied to problems intractable by ordinary computational methods. Since then the interest in "DNA computing" has been growing rapidly, see the list of references in [6]. The impact of the resulting new notions and ideas to the theory of formal languages is visible from the recent Handbook, [8].

A keynote in theoretical studies about DNA computing is a phenomenon known as *Watson-Crick complementarity.* DNA (deoxyribonucleic acid) consists of polymer chains, referred to as *DNA strands.* A chain is composed of *nucleotides* or *bases.* The four DNA bases are customarily denoted by A (adenine), C (cytosine), G (guanine) and T (thymine). A DNA strand can be viewed as a word over the *DNA alphabet* $\Sigma_{DNA} = \{A, C, G, T\}$. The familiar DNA double helix arises by the boundage of two strands. The Watson-Crick complementarity comes into the

*Dedicated to Ferenc Gécseg on his 60th birthday. My "Super-Brother Feri" has been an invaluable companion on the paths of science and a true friend in everyday life. I owe him much and wish him sunny days on the road ahead.

†Academy of Finland and Turku Centre for Computer Science Lemminkäisenkatu 14 A FIN-20520 Turku, Finland, e-mail: asalomaa@utu.fi.

picture in the formation of such *double strands*. The bases A and T are *complementary*, and so are the bases C and G. Bonding occurs only if the bases in the corresponding positions in the two strands are complementary.

Consider the letter-to-letter endomorphism h_W of Σ^*_{DNA} defined by

$$h_W(A) = T,\ h_W(T) = A,\ h_W(G) = C,\ h_W(C) = G.$$

The morphism h_W will be referred to as the *Watson-Crick morhism*. Thus, a DNA strand X bonds with $h_W(x)$ to form a double strand. (We ignore here the orientation of the strands, indicated customarily by speaking of the 5′- and 3′-ends of a strand. We also would like to point out that we use the nowadays standard term "DNA computing" although, in our estimation, "DNA-based computing" would be more appropriate.) The complementarity of two strands leads (under appropriate conditions) to bondage. By encoding information on the original strands in a clever way, far-reaching conclusions can be made from the mere fact that bondage has occurred. This means that the phenomenon of complementarity provides computing power. The idea of using the fundamental knowledge, concerning how the double strands have possibly come into being, is central in Adleman's experiment, [1]. The idea is also behind the computational universality of many models of DNA computing, [9], [6].

Complementarity can be viewed also as a language-theoretic operation. As such h_W is only a morphism of a special kind. However, the operational complementarity can be considered also as a tool in a developmental model: undesirable conditions in a string *trigger* a transition to the complementary string. Thus, the class of "bad" strings is somehow specified. Whenever a bad string x is about to be produced by a generative process, the string $h_W(x)$ is taken instead of x. If the generative process produces a unique sequence of strings (words), the sequence continues from $h_W(x)$. The class of bad strings has to satisfy the following *soundness condition*: whenever x is bad, the complementary string $h_W(x)$ is not bad. This condition guarantees that no bad strings are produced.

While the operational complementarity can be investigated in connection with any generative process for words, it seems particularly suitable for *Lindenmayer systems*, the systems themselves being developmental models. The simplest L system, namely the DOL system, has been thoroughly investigated, [7]. A DOL system generates a sequence of words. When it is augmented with a trigger for complementarity transitions, as described above, the resulting sequences contain no bad words. The study of such "Watson-Crick DOL systems" was begun in [4] and [5], and will be continued in the present paper. The present paper is largely self-contained. In particular, no knowledge of [4] and [5] is required on the part of the reader. For more information about formal languages, L systems or DNA computing, the references [10], [7] or [6], respectively, may be consulted.

The formal definitions will be given below. An important remark should be made already at this stage. So far we have spoken only of the four-letter DNA alphabet but in our theoretical considerations below the size of the alphabet will

be arbitrary. Indeed, we will consider *DNA-like alphabets*

$$\Sigma_n = \{a_1, \ldots, a_n, \bar{a}_1, \ldots, \bar{a}_n\} \quad (n \geq 2)$$

and refer to the letters a_i and $\bar{a}_i$, $i = 1, \ldots, n$, as *complementary*. The endomorphism h_W of Σ_n^* defined by

$$h_W(a_i) = \bar{a}_i, \; h_W(\bar{a}_i) = a_i, \; i = 1, \ldots, n,$$

is also now referred to as the *Watson-Crick morphism*. When we view the original DNA alphabet in this way, the association of letters is as follows:

$$a_1 = A, \; a_2 = G, \; \bar{a}_1 = T, \; \bar{a}_2 = C.$$

(Observe that this conforms with the two definitions of h_W.) The nucleotides A and G are *purines*, whereas T and C are *pyrimidines*. This terminology is extended to concern DNA-like alphabets: the non-barred letters $a_1, \ldots, a_n$ are called *purines*, and the barred letters $\bar{a}_1, \ldots, \bar{a}_n$ are called *pyrimidines*. The class of bad words, considered most frequently in the sequel, consists of words where the pyrimidines form a majority.

In spite of their formal simplicity, Watson-Crick DOL systems have quite remarkable properties. This observation made already in [4] and [5] will be further substantiated in this paper. In particular, we will be concerned with basic decision problems. The following decision problem turns out to be very significant.

Problem Z_{pos}. Decide whether or not a negative number appears in a given Z-rational sequence of integers.

The decidability status of Z_{pos} is open, although the problem is generally believed to be decidable. The input is of course assumed to be given by some effective means such a linear recurrence with integer coefficients, or a square matrix M with integer entries such that the sequence is read from the upper right corners of the powers M^i, $i = 1, 2, 3, \ldots$. Further discussion about this problem and its different representations can be found in [3] and [7].

Ordinary DOL systems have been widely investigated and their properties are fairly well understood, whereas rather little is known about Watson-Crick DOL systems. It was already observed in [5] that graphs associated to them, as well as certain paths in such graphs, are very useful for studying the systems. Such "Watson-Crick walks and roads" will be investigated in this paper from a more general point of view.

2 Graphs associated to DOL systems

We will use standard language-theoretic notation. In particular, λ is the empty word, $|w|$ is the length of the word w, and $|w|_a$ (resp. $|w|_\Sigma$) is the number of occurrences of a (resp. letters of Σ) in w. The minimal alphabet of a word w is denoted by $alph(w)$.

An equivalence relation $\equiv$ on Σ^* is called a *morphic equivalence* if it preserves all endomorphisms of Σ^*, that is, whenever h is an endomorphism of Σ^* and $x \equiv y$ then also $h(x) \equiv h(y)$. Typical examples of morphic equivalences are:

(i) $alph(x) = alph(y)$,

(ii) x and y are powers of the same primitive root,

(iii) x and y have the same Parikh vector.

Of (i)–(iii), only (i) is of finite index. Note also that the equivalence defined by $|x| = |y|$ is not morphic.

A *DOL system* is a triple $G = (\Sigma, g, w_0)$, where Σ is an alphabet, $w_0 \in \Sigma^*$ (the *axiom*) and g is an endomorphism of Σ^*. (In the sequel g is often defined in terms of *productions*, indicating the image of each letter.) A DOL system defines the *sequence* $S(G)$ of words w_i, $i \geq 0$, where $w_{i+1} = g(w_i)$, for all $i \geq 0$. It defines also the *language* $L(G)$, consisting of all words in $S(G)$, the *length sequence* $|w_i|$, $i \geq 0$, as well as the *growth function* $f(i) = |w_i|$.

Given a DOL system $G = (\Sigma, g, w_0)$ and a morphic equivalence $\equiv$ on Σ^*, the *associated graph* $H(G, \equiv)$ is defined as follows. As a preparation for the sequel, we give this simple definition inductively, denoting the equivalence class of a word w by $[w]$. First the initial node of H, labeled by the equivalence class $[w_0]$, is created. Whenever a node labeled by $[w_i]$ has been created but no node labeled by $[g(w_i)]$ has been created, then the latter node is created and an arrow labeled by 0 is drawn from the former to the latter node. If the node labeled by the equivalence class $[g(w_i)]$ has already been created and denoted by, say, $[w_j]$ then an arrow labeled by 0 is drawn from the node $[w_i]$ to the node $[w_j]$.

Thus, all arrows (edges) in the (di)graph H are labeled by 0. (This is because H is a special case of the definition in the next section, where two labels are needed for the arrows.) The graph H is infinite if all words in $S(G)$ belong to different equivalence classes of $\equiv$. Starting from the initial node, an i-step *walk* (path) ends at a node labeled by the equivalence class $[w_i]$, where w_i is the ith word in the sequence $S(G)$. If $\equiv$ is of finite index, the digraph H is finite and can be separated into an "initial mess" and a "loop" in the customary fashion. This fact can also be expressed as the following theorem.

Theorem 2.1 *Let G be a DOL system and $\equiv$ a morphic equivalence (with the same alphabet) of finite index. Then the equivalence classes represented by the words in $S(G)$ form an ultimately periodic sequence.*

Proof. The claim follows by the construction of the graph H. Since $\equiv$ is of finite index, some word in the sequence $S(G)$, say $w_i = g(w_{i-1})$, represents an equivalence class already represented by w_j, for some $j < i$. If i has its smallest possible value, the words $w_0, \ldots, w_{j-1}$ represent equivalence classes in the "initial mess" and the words $w_j, \ldots, w_i$ equivalence classes in the "loop" of the ultimately periodic sequence. □

Theorem 2.1 is a general formulation of many known periodicity results concerning DOL sequences. For instance, the alphabets $alph(w_i)$ and prefixes or suffices of fixed lengths form ultimately periodic sequences, [7]. Such results are immediate corollaries of Theorem 2.1.

Observe that the graph H may be finite although $\equiv$ is of infinite index. For instance, assume that $x \equiv y$ iff x and y are powers of the same primitive root and that the DOL system G is determined by the axiom ab and prodution $a \to aba$, $b \to a$. Then the graph $H(G, \equiv)$ consists of only one node because all words in the sequence $S(G)$ are powers of the word ab.

Assume now that the alphabet Σ of the given DOL system $G = (\Sigma, g, w_0)$ actually is a DNA-like alphabet, $\Sigma = \Sigma_n$, and that the Watson-Crick morphism h_W is defined as in Section 1. (Observe that some letters of Σ_n might not occur in $S(G)$.) Then we define the *Watson-Crick graph* $H_W(G, \equiv)$ associated to G and a morphic equivalence $\equiv$ as follows. We are now dealing with two morphisms: g and the composition $h_W g$ (meaning that first g, then h_W is applied). The edge labels 0 and 1, respectively, are associated to these morphisms, respectively.

To construct $H_W(G, \equiv)$, we again first create the *initial node* labeled by $[w_0]$. Assume that a node labeled by $[w]$ has already been created and no node labeled by $[g(w)]$ (resp. $[h_W g(w)]$) has been created, then the latter node is created and an arrow labeled by 0 (resp. by 1) is drawn from the node $[w]$ to the newly created node. If a node labeled by $[g(w)]$ (resp.$[h_W g(w)]$) has been created and denoted by, say, $[w']$ (resp. $[w'']$) then an arrow labeled by 0 (resp. by 1) is drawn from the node $[w]$ to $[w']$ (resp. to $[w'']$).

Thus, H_W is a (possibly infinite) (di)graph, where exactly two arrows emanate from each node. A sufficient but not necessary condition for the finiteness of H_W is that the morphic equivalence $\equiv$ is of finite index. The smallest possible graph H_W consists of a single node $[w_0]$ with two arrows, labeled by 0 and 1, emanating from and going into $[w_0]$. This situation arises, for instance, if $x \equiv y$ is defined by the condition $alph(x) = alph(y)$, and every letter of Σ occurs in every word of $S(G)$. (It is easy to verify that in such a case an application of $h_W g$ never leads to smaller alphabets.) This smallest possible graph $H_W(G, \equiv)$ is referred to as *trivial.*

A *walk* W in the graph $H_W(G, \equiv)$ is any finite path beginning from the initial node. Walks in graphs $H(G, \equiv)$ can be described (equivalently) either by the sequence of nodes or by the sequence of edges because there is only one possibility at each node. In graphs $H_W(G, \equiv)$ there are two possibilities, perhaps both leading to the same node (as is the case, for instance, with the trivial graph). Consequently, walks in H_W must be described by listing the sequence of *edges* visited. In this fashion, we get a one-to-one correspondence between walks in H_W and words over the binary alphabet $\{0, 1\}$.

Many types of questions can be asked concerning the notions introduced in this paper – we will focus on some *decision problems.* Therefore, the following *general observation* is significant. Consider decision problems (for instance, the emptiness problem) involving languages of the form $L \cap K$, where L is a DOL language and K is in one of the classes of the Chomsky hierarchy. Such problems are usually decidable (resp. undecidable) if K ranges over regular (resp. context-sensitive) languages.

If K ranges over context-free languages, the decidability is often hard to settle, although intuitively the problem might seem to be decidable. Such problems are often algorithmically equivalent to the problem Z_{pos}.

3 Watson-Crick DOL systems

Consider again a DNA-like alphabet Σ_n and the Watson-Crick morphism h_W. A *trigger* TR is any recursive subset of Σ_n^* satisfying the following condition: whenever x is in TR, then $h_W(x)$ is in the complement of TR, that is, in $\Sigma_n^* - TR$.

According to our terminology in the Introduction, TR consists of "bad" strings. The restriction imposed on TR is our *soundness condition*: no "bad" strings result if emerging "bad" words are always replaced by their complementary ones. We now come to our central definitions.

A *Watson-Crick DOL system* is a construct

$$G_W = (G,\ TR),$$

where $G = (\Sigma_n,\ g,\ w_0)$ is a DOL system, TR is a trigger and $w_0 \in \Sigma_n^* - TR$. The *sequence* $S(G_W)$, consisting of words w_i, $i = 0,\ 1,\ \ldots$, is defined by the condition

$$w_{i+1} = \begin{cases} h_w(g(w_i)) & \text{if } g(w_i) \in TR, \\ g(w_i) & \text{otherwise,} \end{cases}$$

for all $i \geq 0$. The *language*, *length sequence* and *growth function* of G_W are defined for $S(G_W)$ as for ordinary DOL systems.

The *Watson-Crick graph* $H_W(G_W,\ \equiv)$ associated to a Watson-Crick DOL system $G_W = (G,TR)$ and morphic equivalence $\equiv$ equals, by definition, the Watson-Crick graph $H_W(G, \equiv)$. (Thus, $H_W(G_W,\ \equiv)$ is independent of the trigger.) A *Watson-Crick walk* $WW(G_W,\ \equiv)$ associated to G_W and $\equiv$ is the walk in $H_W(G_W,\ \equiv)$ determined by the binary word $u_1 \ldots u_k$ such that, for $1 \leq i \leq k$, $u_i = 0$ (resp. $u_i = 1$) if $w_i = g(w_{i-1})$ (resp. $w_i = h_W(g(w_{i-1}))$) in $S(G_W)$.

Thus, the binary word $u_1 \ldots u_k$, determining the sequence of *edges* visited, is actually independent of the equivalence $\equiv$. If we are only interested in the sequence of edges, we may speak of the Watson-Crick walk of G_W, without specifying the equivalence. The latter becomes important if we are interested in the sequence of *nodes* visited. Observe that, viewed as a sequence of edges, the Watson-Crick walk in the trivial graph can be quite complicated. This is exemplified in Theorem 3.3 below. The next theorem follows directly from the definitions.

Theorem 3.1 *Viewed as binary words, all Watson-Crick walks* $WW(G_W,\ \equiv)$ *are prefixes of the same infinite (binary) word* $WW(G_W)$. *Thus, each Watson-Crick walk of* G_W *is completely determined by its length.* □

The infinite word $WW(G_W)$ is called the *Watson-Crick road* of G_W. Two Watson-Crick DOL systems are called *road equivalent* if they have the same Watson-Crick road. A Watson-Crick DOL system G_W is called *stable* if its Watson-Crick road equals 0^ω (that is, the infinite word consisting of 0s).

Thus, the Watson-Crick road completely characterizes the complementarity transitions: letters 1 occur in positions such that a transition takes place at the corresponding step. A system being stable means that no complementarity transitions occur, that is, the sequence is obtained as in an ordinary DOL system. The *stability problem* is basic in the study of Watson-Crick DOL systems. In general, the problem is undecidable. We have the following more specific results.

Theorem 3.2 *The stability problem is decidable for Watson-Crick DOL systems with regular trigger but undecidable for systems with context-sensitive triggers.*

Proof. A Watson-Crick DOL system $G_W = (G, TR)$ is stable iff the intersection $L(G) \cap TR$ is empty, where $L(G)$ is the language of G, viewed as an ordinary DOL system. But the emptiness of such an intersection is decidable (resp. undecidable) for regular (resp. context-sensitive) triggers TR, see [7] (resp. [2]). □

We now show that Watson-Crick roads can be arbitrarily complex, even if attention is restricted to systems whose Watson-Crick graph is trivial. Let φ be a recursive function mapping the set of positive integers into $\{0, 1\}$. We denote by u_φ the infinite binary word whose ith letter equals 1 exactly in case $\varphi(i) = 1$, for all $i \geq 1$.

Theorem 3.3 *For every recursive function φ, a Watson-Crick DOL system whose Watson-Crick road equals u_φ can be effectively constructed. Moreover, the items involved can always be chosen in such a way that the morphic equivalence is defined by the relation $alph(x) = alph(y)$ and that the associated Watson-Crick graph is trivial.*

Proof. Given φ, we construct a Watson-Crick DOL system $G_W = (G, TR)$ as follows. The alphabet of the DOL system G is $\Sigma_2 = \{a_1, a_2, \bar{a}_1, \bar{a}_2\}$. We prefer to write Σ_2 as the original DNA alphabet $\{A, T, C, G\}$ in the way indicated in Section 1 (trusting that the slight notational ambiguity causes no confusion). The axiom of the system is $ACGT$, and the morphism g is defined by the rules

$$A \to A, \quad C \to C^2, \quad G \to G^2, \quad T \to T.$$

The morphic equivalence is defined by the condition: $x \equiv y$ iff $alph(x) = alph(y)$. Then (independently of TR which we have not yet defined) the Watson-Crick graph $H_W(G_W, \equiv)$ is trivial. This follows because each word in the sequence $S(G_W)$ equals either $w_1(i) = AC^{2^i}G^{2^i}T$ or $w_2(i) = TG^{2^i}C^{2^i}A$, for some i.

We now define the trigger by

$$TR = \{w_1(i), w_2(i) | i \in \varphi^{-1}(1)\}.$$

Clearly, TR is recursive. It is also easy to verify that this construction satisfies the theorem. □

A very natural trigger (and the only one considered in [5]) is the set of words, where the pyramidines (barred letters) are in strict majority. Thus, we denote $\Sigma_{pyr} = \{\bar{a}_1, \ldots, \bar{a}_n\}$ and consider the language

$$PYR = \{w \in \Sigma_n^* \mid |w|_{\Sigma_{pyr}} > \frac{|w|}{2}\}.$$

Clearly, PYR satisfies the soundness condition. Watson-Crick DOL systems (G, PYR) will be referred to as *standard*. Consequently, for a standard system G_W, in every word of $S(G_W)$ there are at least as many purines as pyrimidines.

Observe that PYR and its complement are context-free nonregular languages. Thus, considering Theorem 3.2, we are closer to the borderline between decidability and undecidability. Indeed, the following result was established in [5].

Theorem 3.4 *The stability problem for standard Watson-Crick DOL systems is algorithmically equivalent to the problem* Z_{pos}. □

An infinite binary word is referred to as *ultimately periodic* if it is of the form uv^ω, where $u \in \{0, 1\}^*$ and $v \in \{0, 1\}^+$. The trigger used in the general result Theorem 3.3 is very complicated. However, all ultimately periodic roads can be realized with simpler triggers.

Theorem 3.5 *Every ultimately periodic Watson-Crick road can be expressed in the form* $WW(G_W)$ *where* G_W *is standard (resp.* G_W *has a finite trigger).*

Proof. Assume uv^ω is the given word, where

$$u = b_1 \ldots b_k,\ v = b_{k+1} \ldots b_l,\ k \geq 0,\ l \geq k+1,\ b_j \in \{0, 1\}.$$

We construct a Watson-Crick DOL system $G_W = (G, TR)$. The alphabet of G equals $\{a_0, \ldots, a_l, \bar{a}_0, \ldots, \bar{a}_l\}$, the axiom is a_0 and the morphism is defined by the productions

$a_j \to a_{j+1}$ or $a_j \to \bar{a}_{j+1}$, depending whether $b_{j+1} = 0$ or $b_{j+1} = 1$, $0 \leq j \leq l-1$;
$a_l \to a_{k+1}$ or $a_l \to \bar{a}_{k+1}$, depending whether $b_{k+1} = 0$ or $b_{k+1} = 1$;
$\bar{a}_j \to \bar{a}_j,\ 0 \leq j \leq l$.

If we now choose $TR = PYR$ or $TR = \{\bar{a}_1, \ldots, \bar{a}_l\}$, the resulting system G_W has the Watson-Crick road uv^ω. □

For any Watson-Crick DOL system G_W and any morphic equivalence $\equiv$, every node in the graph $H_W(G_W, \equiv)$ is *reachable* in the sense that there is a walk ending with that node. This follows by the construction of H_W. However, the Watson-Crick road of G_W does not necessarily pass through all nodes of G_W.By the *reachability problem* we understand the problem of deciding, given G_W, $\equiv$ and a node N in H_W, whether or not the Watson-Crick road of G_W passes through N.

In this context we assume that the morphic equivalence is defined by the condition $alph(x) = alph(g)$, implying that H_W is finite. Examples can be given of cases, where N actually is reachable but the shortest prefix of the Watson-Crick road of G_W ending with N is very long (for instance, in terms of the number of nodes of H_W). This is natural, in view of the following theorem.

Theorem 3.6 *The reachability problem is undecidable. For standard Watson-Crick DOL systems G_W, any algorithm for solving the reachability problem can be converted into an algorithm for solving the problem Z_{pos}.*

Proof. The second sentence has been established in [5]. To prove the first sentence, we show that the decidability of the problem would imply the decidability of the emptiness problem for languages $L(G) \cap K$, where K is context-sensitive and G is a DOL system. However, the latter problem is known to be undecidable.

Given K and G over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$, we construct a Watson-Crick DOL system $G_W = (G', K)$. (Without loss of generality, we assume that the language K does not contain λ.) The DOL system G' is almost the same as G; we only extend the alphabet Σ to the DNA-like alphabet Σ_n and add the productions $\bar{a}_i \to \bar{a}_i$, $1 \leq i \leq n$. Clearly, K satisfies the soundness condition for triggers because, for $x \in K$, $h_W(x)$ consists of pyramidines and K contains no such words.

Two possibilities arise. If $L(G) \cap K = \phi$ then $S(G_W) = S(G)$ because no complementarity transition takes place. If $L(G) \cap K \neq \phi$ and w_i is the first word in $S(G)$ belonging to K (we may assume that w_i is not the axiom), then $S(G_W)$ coincides with $S(G)$ up to w_{i-1}, after which $S(G_W)$ consists of only repetitions of $\bar{w}_i$. Because of our agreement concerning the morphic equivalence, the latter alternative occurs exactly in case the Watson-Crick road of G_W passes through some node in H_W labeled by pyramidines. This is a question we can settle if we can decide the reachability problem. □

We conclude this section with an example of a standard Watson-Crick DOL system G_W, due originally to [4]. The alphabet of G_W is Σ_3, the axiom is $a_1 a_2 \bar{a}_3$, and the productions are

$$a_1 \to a_1,\ a_2 \to a_2,\ a_3 \to a_3,\ \bar{a}_1 \to \bar{a}_1\bar{a}_2,\ \bar{a}_2 \to \bar{a}_2,\ \bar{a}_3 \to \bar{a}_3^3.$$

The graph $H_W(G_W, \equiv)$ where $\equiv$ is again defined as in Theorem 3.6, consists of two nodes: the initial node N_1 labeled by $\{a_1, a_2, \bar{a}_3\}$, and the node N_2 labeled by $\{\bar{a}_1, \bar{a}_2, a_3\}$. The arrows labeled by 0 preserve both nodes, whereas the arrows labeled by 1 interchange them. The Watson-Crick road of G_W begins with the word $10110^5110^{17}11$. In general, there is an exponentially growing sequence of 0s between words 11. Explicitly, after the first position the bit 1 occurs exactly in positions $3^{i+1} + i$ and $3^{i+1} + i + 1$, for all $i \geq 0$. It is interesting to note that only three of the four arrows of H_W are used on the Watson-Crick road; the arrow from N_1 to itself is never used. The example shows the validity of the following theorem.

Theorem 3.7 *The Watson-Crick road of a standard system is not necessarily ultimately periodic.* □

Theorem 3.7 should be contrasted to the many results about context-free language showing ultimate periodicity; recall that the trigger in a standard system is context-free.

The growth function of the system G_W fluctuates between a linear function (due to the productions $\bar{a}_1 \to \bar{a}_1\bar{a}_2$, $\bar{a}_2 \to \bar{a}_2$) and an exponential function (due to $\bar{a}_3 \to \bar{a}_3^3$). Such a fluctuation is not possible for DOL growth functions. Indeed, it is shown in [4] that the growth function of G_W is not Z-rational. The system G_W is the smallest standard system with these properties (strange growth function and nonperiodicity) we have been able to find. It would be interesting to have similar examples with the original DNA alphabet or, equivalently, Σ_2.

4 Equivalence problems

The decidability of various *equivalence problems* constitutes a central chapter in the history of L systems, see [7]. We use here the standard terminology. Thus, the *sequence* (resp. *growth*) *equivalence problem* for DOL systems consists of deciding of two given DOL systems whether or not they generate the same sequence (resp. growth function). For DOL systems, the decidability of the growth equivalence problem was settled first. It was also shown quite early, that the decidability of the sequence equivalence implies the decidability of the language equivalence and vice versa, whereas the decidability itself remained as a celebrated open problem, until it was finally settled in the late 70s, see [7].

Clearly, the *sequence*, *language* and *growth equivalence problems* can be formulated for Watson-Crick DOL systems exactly as for ordinary DOL systems. In addition, we have the very natural *road equivalence problem* for Watson-Crick DOL systems: given two systems, decide whether or not they define the same Watson-Crick road. Thus, the road equivalence of two Watson-Crick DOL systems means only that the complementarity transitions occur in the two sequences at the same steps; the two sequences themselves can be very different. For instance, two stable systems are always road equivalent.

Thus, road equivalence does not imply sequence, language or growth equivalence. On the other hand, sequence equivalence (which implies language and growth equivalence) does not imply road equivalence. For instance, consider two standard Watson-Crick DOL systems G_1 and G_2 over the DNA alphabet. The axiom of both systems is AG. The productions in G_1 are

$$A \to T, \quad G \to C, \quad C \to C^2, \quad T \to T^2,$$

whereas in G_2 they are

$$A \to A, \quad G \to G, \quad C \to C^3, \quad T \to T^3.$$

Then $S(G_1) = S(G_2)$ but G_1 and G_2 are not road equivalent. In fact, the Watson-Crick roads of G_1 and G_2 are 1^ω and 0^ω, respectively. (Observe that it is irrelevant in both systems how we choose the productions for C and T.)

It is also possible that two systems are sequence equivalent when viewed as ordinary DOL systems but not as Watson-Crick DOL systems, and vice versa. We have seen that the above systems G_1 and G_2 are sequence equivalent when viewed as standard (implying that $TR = PYR$) Watson-Crick DOL systems. They are clearly not sequence equivalent when viewed as ordinary DOL systems: after the axiom the two sequences differ at every step, and even the word lengths differ from the third word on. On the other hand, consider two systems G_3 and G_4 with the axiom A, where the productions in G_3 are

$$A \to CTC, \quad C \to CTC, \quad T \to \lambda, \quad G \to G^3,$$

and those in G_4 are

$$A \to CTC, \quad C \to C, \quad T \to TCCT, \quad G \to G^4.$$

Viewed as ordinary DOL systems, G_3 and G_4 are sequence equivalent. Indeed, $S(G_3) = S(G_4)$ begins with A, followed by the words $(CTC)^{2^i}$, $i \geq 0$. If G_3 is viewed as a standard Watson-Crick DOL system, the sequence $S(G_3)$ consists of the words

$$A, \quad GAG, \quad G^3CTCG^3, \quad G^{3^{i+1}}(CTC)^{2^i}G^{3^{i+1}}, \quad i \geq 1.$$

If G_4 is viewed similarly, the sequence $S(G_4)$ consists of the words

$$A, \quad GAG, \quad G^4CTCG^4, \quad G^{4^{i+1}}(CTC)^{2^i}G^{4^{i+1}}, \quad i \geq 1.$$

Consequently, G_3 and G_4 are not sequence equivalent. Observe, however, that G_3 and G_4 are road equivalent: both define the Watson- Crick road 10^ω.

The above examples serve the purpose of illustrating the great variety of problems of new types brought forward by the different notions of equivalence. Indeed, some challenging decision problems in this area remain open. According to the general observation made at the end of Section 2, it is to be expected that equivalence problems involving arbitrary (resp. regular) triggers are undecidable (resp. decidable). The case of arbitrary (context-sensitive) triggers will be dealt with in Theorem 4.1, whereas we hope to return to regular triggers in a forthcoming paper. Equivalence problems involving context-free triggers (as is the case with standard systems) are very challenging. Intuitively, the problems seem to be decidable. But, as shown in Theorem 4.2, they are at least as hard as the problem Z_{pos}.

Theorem 4.1 *The road, growth, sequence and language equivalence problems are all undecidable for Watson-Crick DOL systems with context-sensitive triggers.*

Proof. We use again the undecidability of the emptiness of the intersection $L(G) \cap K$, where G is an ordinary DOL system and K is a context-sensitive language. Indeed, the argument is similar to the one used in the proof of Theorem 3.6.

Assume that we are given a DOL system G and a context-sensitive language K over the alphabet Σ. Without loss of generality, we assume that $L(G)$ is infinite and that the axiom of G is not in K. We can also find a word $u \in \Sigma^+$ not in $L(G)$.

We now construct two Watson-Crick DOL systems G' and G'' by taking the axiom of G, extending the alphabet Σ to a DNA-like alphabet Σ_n by adding to Σ the barred version $\bar{a}$ of each letter a and, finally, by adding to G all productions $\bar{a} \to \bar{a}$, where $a \in \Sigma$. The systems $G'$and G'' are identical except that G' has the trigger $\{u\}$, whereas K is the trigger of G''.

It is now easy to verify, exactly as in the proof of Theorem 3.6, that $G'$and G'' are road, growth, sequence or language equivalent exactly in case $L(G) \cap K = \phi$. Indeed, the Watson-Crick road of G' equals 0^ω, and the growth function, sequence and language of G' coincide with that of G. Each of these statements holds for G'' exactly in case G'' is stable, that is, $L(G) \cap K = \phi$. (Our assumption concerning the infinity of $L(G)$ is needed to justify this conclusion for growth functions.) □

Theorem 4.2 *Any algorithm for solving the road, growth, sequence or language equivalence problem for standard Watson-Crick DOL systems can be converted into an algorithm for solving the problem Z_{pos}.*

Proof. By Theorem 3.4, it suffices to show that an algorithm for solving any of the four equivalence problems for standard Watson-Crick DOL systems can be converted into an algorithm for solving the stability of standard Watson-Crick DOL systems.

Thus, we have to decide whether or not a given Watson-Crick DOL system $G_W = (G, PYR)$ is stable, where $G = (\Sigma_n, g, w_0)$ is a DOL system. Here Σ_n is a DNA-like alphabet and, thus, consists of $2n$ letters. We now extend Σ_n to a DNA-like alphabet $\Sigma_{2n} = \Sigma_n \cup \bar{\Sigma}_n$, where $\bar{\Sigma}_n$ consists of barred versions of letters of Σ_n. (The new bars should not be confused with those appearing in letters of Σ_n.) In connection with Σ_{2n}, the letters of $\bar{\Sigma}_n$ are considered pyrimidines and the set $PYR \subseteq \Sigma_{2n}^*$ is defined accordingly. Consider also the extension g' of g to Σ_{2n}^*, where $g'(a) = a$ for all $a \in \bar{\Sigma}_n$, and define the standard Watson-Crick DOL system $G'_W = (G', PYR)$, where $G' = (\Sigma_{2n}, g', w_0)$ and PYR is defined in connection with Σ_{2n}. Clearly, G'_W is stable and defines the Watson-Crick road 0^ω. Consequently, G_W is stable exactly in case G_W and G'_W are road, sequence or language equivalent. This means that an algorithm for deciding one of these three equivalence problems decides also the stability problem.

The same conclusion cannot be made directly as regards the growth equivalence problem: it is conceivable that G_W and G'_W are growth equivalent although complementarity transitions take place in G_W. However, the proof of Theorem 3.4 in [5] is easily modified to exclude this possibility. In this proof, the Z-rational sequence (given for the problem Z_{pos}) was expressed as the difference of two DOL length sequences, generated by systems both having n letters. When the systems are run simultaneously and the letters of the original systems are viewed as purines and pyrimidines, the combined system is stable exactly in case the Z-rational sequence assumes never a negative value. We now consider two new letters a_{n+1} and $\bar{a}_{n+2}$, as well as their complementary ones $\bar{a}_{n+1}$ and a_{n+2}. The axiom of the combined system is catenated with the word $a_{n+1}^i \bar{a}_{n+2}^i$, where the new letters have the productions $a_{n+1} \to a_{n+1}^j$ and $\bar{a}_{n+2} \to \bar{a}_{n+2}^j$. Here i and j are large enough

for the new letters to dominate the growth function of the combined system. (The new letters contribute the same number of purines and pyrimidines and, thus, do not affect membership in PYR.) The choice $\bar{a}_{n+1} \to \bar{a}_{n+1}^{2j}$, $a_{n+2} \to a_{n+2}^{2j}$ now guarantees that a complementarity transition changes the growth function. □

We do not know whether Theorem 4.2 holds in the reverse order, that is, whether an algorithm for solving the problem Z_{pos} can be converted into an algorithm for solving the equivalence problems.

5 Conclusion

In Watson-Crick DOL systems one investigates a classical topic, Lindenmayer systems, from the new angle provided by the idea of complementarity in DNA computing. In this paper we have focused our attention to the fundamental information brought forward by the associated Watson-Crick road. Many problems in this area are very challenging, especially because of their interconnection with some celebrated open problems. It would be a fascinating project to apply DNA computing itself towards the solution of these problems.

References

[1] Adleman, L. Molecular computation of solutions to combinatorial problems. *Science* 266, 1994, 1021–1024.

[2] Harrison, J. Morphic congruences and DOL languages. *Theoretical Computer Science* 134, 1994, 537-544.

[3] Kuich, W. and Salomaa, A. *Semirings, Automata, Languages.* Springer-Verlag, Berlin, Heidelberg, New York, 1986.

[4] Mihalache, V. and Salomaa, A. Lindenmayer and DNA: Watson-Crick DOL systems. *EATCS Bulletin* 62, 1997, 160–175.

[5] Mihalache, V. and Salomaa, A. Language-theoretic aspects of DNA complementarity. Submitted for publication.

[6] Paun, G., Rozenberg, G. and Salomaa, A. *DNA Computing - New Computing Paradigms.* Springer-Verlag, Berlin, Heidelberg, New York, 1998.

[7] Rozenberg, G. and Salomaa, A. *The Mathematical Theory of L Systems.* Academic Press, New York, 1980.

[8] Rozenberg, G. and Salomaa, A. (eds) *Handbook of Formal Languages*, Vol. 1–3. Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[9] Salomaa, A. Turing, Watson-Crick and Lindenmayer. Aspects of DNA complementarity. In Calude, C., Casti, J. and Dinneen, M. (eds) *Unconventional Models of Computation.* Springer-Verlag, Singapore, 1998, 94–107.

[10] Salomaa, A. *Formal Languages.* Academic Press, New York, 1973.

Acta Cybernetica **14** (1999) 193-203.

Generalized fairness and context-free languages

Kai Salomaa* Sheng Yu*†

Abstract

The notion of fairness for generalized shuffle operations was introduced in [10]. The n-fairness property requires, roughly speaking, that in any prefix of a word the difference of the numbers of occurrences of two symbols is at most n. Here we give a new simplified proof for the decidability of uniform n-fairness for context-free languages. Also, we show that the more general, linear or logarithmic, fairness notions are decidable.

1 Introduction

Fairness constraints can be used to restrict the behavior of concurrent processes. For a state sequence to be fair, a minimal requirement is that a process that is enabled infinitely often will occur infinitely often. Many different notions of fairness have been studied in modeling concurrency [3], some recent references are [2, 5, 14]. Fairness of automata on infinite objects and of cooperating grammar systems is considered in [8, 13, 15].

Questions of fairness in formal language theory were initiated by the study of trajectories [9, 10]. A trajectory is a word over a two-letter alphabet that is used to "control" the shuffle operation on given words. Trajectories yield very general operations of parallel composition of words and languages and have applications in the parallelization of languages, in representing a language in terms of simpler components.

A trajectory $t \in \{b, c\}^*$ is said to be n-fair if the difference in the number of occurrences of the symbols b and c in any prefix of t is at most n. The shuffle of two words that is controlled by an n-fair trajectory satisfies thus the property that at any point neither word is more than "n steps ahead". It is easy to see that for a context-free set of trajectories T we can effectively decide whether or not T is n-fair. The uniform fairness question asks, for a given set of trajectories T, whether or not there exists an integer n such that T is n-fair.

*Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada, Email: {ksalomaa, syu}@csd.uwo.ca

†Research supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0041630.

It was shown in [11] that also the uniform fairness property is decidable for context-free languages. The proof establishes that for a given context-free grammar G there exists a constant m_G such that the uniform fairness of $L(G)$ can be verified by checking only the derivation trees of height at most m_G. The resulting algorithm is not efficient since it uses an exhaustive search of an exponential number of derivation trees (in terms of the size of the grammar). Furthermore, the argument used to establish the existence of the constant m_G uses fairly complicated operations on the derivation trees.

Here we give a new proof for the decidability of uniform fairness for context-free languages. The fairness property for a context-free language L is determined by considering a regular language that is letter-equivalent to the prefix-language of L and thus it is, essentially, sufficient to decide the property for a regular language. The argument for the correctness of the algorithm is much simpler than in the original proof that relies directly on properties of context-free derivations. Here the adjective "simpler" naturally ignores the fact that we are using the powerful result of Parikh's theorem [12, 4, 16]. Another drawback is that the size of the nondeterministic finite automaton (or regular grammar) obtained by Parikh's theorem to accept a language letter-equivalent to the given context-free language can be much larger than the size of the original context-free grammar. Furthermore, although the algorithm is now conceptually simpler, in the worst case it needs to check an exponential number of cycles in the automaton.

An advantage of the new decidability proof is that exactly the same method allows us to decide the generalized (linear or logarithmic) fairness conditions for context-free languages that were left open in [10].

2 Definitions

Here we present some definitions needed later. More details on formal languages and finite automata can be found e.g. in [1, 4, 16, 17]. For all unexplained notions we refer the reader to these references.

The symbol $\mathbb{N}$ denotes the set of non-negative integers. The cardinality of a finite set S is denoted $\#S$. The set of words over an alphabet Σ is Σ^* and $\Sigma^+ = \Sigma^* - \{\lambda\}$. Here λ denotes the empty word. If not otherwise mentioned, by an alphabet we mean always a finite alphabet. A word w_1 is a *prefix* of $w \in \Sigma^*$ if we can write $w = w_1 w_2$, $(w_1, w_2 \in \Sigma^*)$. For $L \subseteq \Sigma^*$, the prefix-language of L is defined as

$$\mathrm{pref}(L) = \{w \in \Sigma^* \mid (\exists w' \in \Sigma^*)\ ww' \in L\}.$$

The length of a word $w \in \Sigma^*$ is $|w|$ and, for a symbol $c \in \Sigma$, $|w|_c$ denotes the number of occurrences of c in the word w.

Words $w_1, w_2 \in \Sigma^*$ are said to be *letter-equivalent* if for each $c \in \Sigma$ we have $|w_1|_c = |w_2|_c$. Languages L_1 and L_2 $(\subseteq \Sigma^*)$ are *letter-equivalent* if for each $w_1 \in L_1$ there exists $w_2 \in L_2$ such that w_1 and w_2 are letter-equivalent, and vice versa. This means that the words of L_2 are exactly some permutations of the words of L_1.

A *finite automaton* is a four-tuple $A = (Q, \Sigma, s, Q', \delta)$ where Q is the finite set of states, Σ is the input alphabet, $s \in Q$ is the initial state, $Q' \subseteq Q$ is the set of accepting final states, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. Note that for given $q \in Q$ and $a \in \Sigma$ there may exist more than one state q' such that $(q, a, q') \in \delta$, that is, the automaton is allowed to be nondeterministic.

The transition relation is extended in the natural way from symbols of Σ to arbitrary words of Σ^* and we denote also the extended transtion relation ($\subseteq Q \times \Sigma^* \times Q$) by the same symbol δ. The *language accepted by* A is

$$L(A) = \{w \in \Sigma^* \mid (\exists q \in Q')\ (s, w, q) \in \delta\}.$$

A *path* of the automaton A is a sequence

$$\alpha = (q_1, a_1, q_2, a_2, \ldots, a_{m-1}, q_m) \tag{1}$$

where $m \geq 1$, $q_i \in Q$, $i = 1, \ldots, m$, $a_j \in \Sigma$, $j = 1, \ldots, m-1$, and $(q_j, a_j, q_{j+1}) \in \delta$ for all $j \in \{1, \ldots, m-1\}$.

The above path is said to be *accepting* if $q_1 = s$ and $q_m \in Q'$. The automaton A is said to be *reduced* if all states of Q occur in some accepting path of A. It is well known that we can determine the unnecessary states of an automaton and, thus, we can effectively transform A into an equivalent reduced automaton. The *underlying word* of a path α as in (1) is

$$\mathrm{word}(\alpha) = a_1 a_2 \cdots a_{m-1} \in \Sigma^*.$$

A path (1) is said to be a *cycle* if $m \geq 2$ and $q_1 = q_m$. Note that a sequence (q_1) consisting of a single state (with no transitions) is a path but not a cycle. A cycle as in (1) is said to be *primitive* if for no $(i, j) \neq (1, m)$, $1 \leq i < j \leq m$, the sequence

$$(q_i, a_i, q_{i+1}, a_{i+1}, \ldots, a_{j-1}, q_j) \tag{2}$$

is a cycle. A path as in (1) is said to be primitive, if for no $1 \leq i < j \leq m$, the sequence (2) is a cycle.

Intuitively, a primitive cycle does not contain any proper subcycles and a primitive path does not contain any subcycles. In (1) the q_i's need not be distinct and, thus, an automaton A may have an infinite number of cycles (or paths). However, the number of primitive cycles and paths is always finite.

We can define in the natural way the catenation of two paths provided that the first one "ends" with the same state as the second one "begins" with. Let α be as in (1) and

$$\beta = (p_1, b_1, \ldots, b_{r-1}, p_r),$$

where $p_i \in Q$, $b_j \in \Sigma$, $1 \leq i \leq r$, $1 \leq j \leq r-1$. If $q_m = p_1$ then the *catenation* of the paths α and β is defined to be

$$\alpha \cdot \beta = (q_1, a_1, q_2, a_2, \ldots, a_{m-1}, q_m, b_1, p_2, \ldots, b_{r-1}, p_r).$$

If $q_m \neq p_1$ then the catenation of α and β is not defined. Note that a cycle can always be catenated with itself.

Finally we fix the notation used for context-free grammars. A *context-free grammar* is a four-tuple $G = (N, \Sigma, S, P)$, where N is the nonterminal alphabet, Σ is the terminal alphabet, $(N \cap \Sigma = \emptyset)$, $S \in N$ is the initial nonterminal, and $P \subseteq N \times (N \cup \Sigma)^*$ is the finite set of productions. A production $(X, w) \in P$ is denoted as $X \to w$. The productions define in the standard way the rewrite relation of the grammar $\Rightarrow_G \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$, and the language generated by G is $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$.

A context-free grammar $G = (N, \Sigma, S, P)$ is in *Chomsky normal form* if all productions of P are of the form $X \to YZ$ or $X \to b$ where $X, Y, Z \in N$ and $b \in \Sigma$. The grammar G is said to be *regular* (or right-linear) if all productions of P are of the form $X \to wY$ or $X \to w$ where X and Y are nonterminals and w is a terminal string. Every context-free language not containing the empty word can be generated by a grammar in Chomsky normal form. The regular grammars generate exactly the regular languages.

When speaking about the complexity of determining some property of context-free grammars, by the size of the grammar we mean the length of an encoding over a fixed (e.g. a binary) alphabet of the nonterminals, the terminals and the productions of the grammar. Similarly, the size of a finite automaton is determined by an encoding of the states, the input alphabet and the transition relation.

3 Generalized fairness

The n-fairness condition [10, 11] requires that for any distinct symbols b and c and any prefix w' of a given word w, the difference between the numbers of occurrences of b and c in w' is bounded by n. A more general notion of fairness was also informally discussed in [11]. Below we present the more general definition.

Definition 3.1 *Let $g : \mathbb{N} \to \mathbb{N}$ be a function. We say that a language $L \subseteq \Sigma^*$ has the g-fairness property if the following condition holds. For all $b, c \in \Sigma$, if $w = w_1 w_2 \in L$ then*

$$| \, |w_1|_b - |w_1|_c \, | \leq g(|w_1|).$$

The definition requires that the difference between the numbers of occurrences of distinct symbols in any prefix of a word belonging to the language is bounded by the g-image of the length of the prefix. As a special case we get the notion of n-fairness, $n \in \mathbb{N}$, by choosing g to be the function with constant value n.

When we are using a fairness condition as given in Definition 3.1, the function g is referred to as the *fairness function* associated with the condition.

It is a straightforward observation that given a context-free language L and $n \in \mathbb{N}$ the question whether or not L is n-fair is decidable [10]. The *uniform constant-fairness question* asks for a given language L whether or not there exists $n \in \mathbb{N}$ such that L is n-fair. The following result was shown in [11].

Theorem 3.1 *The uniform constant-fairness problem is decidable for context-free languages.*

Below we give a new simplified proof for Theorem 3.1. We will use the following two propositions. A proof of Parikh's theorem can be found for instance in [12, 4, 16]. A more elegant proof using equations over a commutative semigroup is presented in [1, 7].

Proposition 3.1 *(Parikh's Theorem) Each context-free language is letter-equivalent to a regular language. Given a context-free grammar we can effectively construct a regular grammar (or finite automaton) for a letter-equivalent regular language.*

Proposition 3.2 *The prefix-language of a context-free language is context-free.*

Proposition 3.2 follows from the observation that if a language L is generated by a grammar $G = (N, \Sigma, S, P)$ in Chomsky normal form (with possibly an additional production $S \to \lambda$) then $\mathrm{pref}(L)$ is generated by the grammar G' defined as follows. Denote $N' = \{X' \mid X \in N\}$ and let $G' = (N \cup N', \Sigma, S', P \cup P')$ where

$$\begin{aligned} P' \;=\; & \{X' \to YZ',\ X' \to Y' \mid X \to YZ \in P;\ X, Y, Z \in N\} \\ & \cup \{X' \to b \mid X \to b \in P;\ X \in N,\ b \in \Sigma\} \cup \{S' \to \lambda\}. \end{aligned}$$

We assume that S, and hence also S', does not appear in the right side of any production.

Now we can prove Theorem 3.1 relying on the above results.

Proof of Theorem 3.1. Let G be a given context-free grammar with terminal alphabet Σ. By Proposition 3.2 there exists effectively a context-free grammar G' such that $L(G') = \mathrm{pref}(L(G))$. Now there exists $n \in \mathbb{N}$ such that $L(G)$ has the n-fairness property iff there exists $n \in \mathbb{N}$ such that for all $w \in L(G')$,

$$(\forall b, c \in \Sigma) \ \mid |w|_b - |w|_c| \leq n. \tag{3}$$

By Parikh's theorem there exists effectively a finite automaton A such that $L(A)$ is letter-equivalent to $L(G')$. This means that, given $n \in \mathbb{N}$, the condition (3) holds for all $w \in L(G')$ iff the same condition holds for all $w \in L(A)$.

Without loss of generality we can assume that A is reduced. Let $n \in \mathbb{N}$ be fixed. We claim that the condition (3) holds for all $w \in L(A)$ iff the below conditions (i) and (ii) hold.

(i) For all accepting primitive paths α of A, and all $b, c \in \Sigma$,

$$\mid |\mathrm{word}(\alpha)|_b - |\mathrm{word}(\alpha)|_c \mid \leq n.$$

(ii) For all primitive cycles β of A, and all $b, c \in \Sigma$,

$$|\mathrm{word}(\beta)|_b = |\mathrm{word}(\beta)|_c.$$

For the "only if" part assume that (3) holds for all $w \in L(A)$. Consider arbitrary $b, c \in \Sigma$. Now (i) follows from the fact that word$(\alpha) \in L(A)$ if α is an accepting path. For the sake of contradiction assume that

$$\beta = (q_1, a_1, \ldots, q_{m-1}, a_{m-1}, q_1),$$

$q_i \in Q$, $a_j \in \Sigma$, $1 \leq i, j \leq m-1$, $m \geq 2$, is a (primitive) cycle where, for instance,

$$|\text{word}(\beta)|_b > |\text{word}(\beta)|_c.$$

Since A is reduced, the state q_1 is reachable from the initial state and a final state is reachable from q_1. Thus, there exists an accepting path of A that can be written in the form $\gamma_1 \cdot \beta \cdot \gamma_2$. Denote

$$k = |\, |\text{word}(\gamma_1 \cdot \gamma_2)|_b - |\text{word}(\gamma_1 \cdot \gamma_2)|_c \,|.$$

Then

$$\eta = \gamma_1 \cdot \beta^{k+n+1} \cdot \gamma_2$$

is an accepting path such that word(η) does not satisfy (3).

Conversely, assume that the conditions (i) and (ii) hold and let $b, c \in \Sigma$ be arbitrary. Starting from an arbitrary cycle we can delete primitive cycles one-by-one without changing the difference between the numbers of occurrences of the symbols b and c. The process eventually results in a primitive cycle and, thus, it follows that any cycle of A has equally many occurrences of the symbols b and c. An arbitrary accepting path can be written in the form

$$\gamma_1 \cdot \eta_1 \cdot \gamma_2 \cdot \ldots \cdot \eta_{k-1} \cdot \gamma_k,$$

where $k \geq 1$, η_i, $1 \leq i \leq k-1$, is a cycle and $\gamma_1 \cdot \gamma_2 \cdot \ldots \cdot \gamma_k$ is a primitive accepting path. This completes the proof since the words of $L(A)$ are exactly the underlying words of accepting paths of A. □

In the above proof, the answer for a reduced automaton A is "yes" for sufficiently large n iff the condition (ii) holds, that is, the condition (i) would not be needed at all. The condition (i) was included only to make more transparent the idea that the same method can be used below for the linear and logarithmic fairness functions. When considering a fixed fairness function, we need to have some condition also for the accepting primitive paths.

The above proof of Theorem 3.1 is quite simple when compared to the proof given in [11], at least if we ignore the fact that we are relying on Parikh's theorem. On the other hand, the algorithm obtained from the original proof is more efficient, although it also requires exponential time. Note that the construction that for a given context-free grammar G produces a regular grammar generating a language letter-equivalent to $L(G)$ greatly increases the size of the grammar [4, 12, 16]. We do not know whether the construction could be improved in this respect.

The construction using Chomsky normal form grammars outlined above for the proof of Proposition 3.2 also increases the size of a given grammar since the

Chomsky normal form can have many more nonterminals and productions than the original grammar. However, this overhead could be avoided. The construction in the proof of Proposition 3.2 does not require that the grammar is in Chomsky normal form if one uses more carefully defined rules to determine the behavior of the primed nonterminal that denotes the end of the prefix in the derivation tree. Also, the assumption that the given finite automaton is reduced is not problematic since this property can be tested in low polynomial time.

Although it is perhaps intuitively simpler than the construction of [11] that determines properties of context-free derivations, the algorithm testing the primitive loops of a finite automaton still requires exponential time. This follows from the below example.

Example 3.1 *Let $n \in \mathbb{N}$ and consider the finite automaton $A_n = (Q, \Sigma, s, Q', \delta)$ where $\Sigma = \{a, b\}$, $Q = \{1, 2, \ldots, n\}$, $s = 1$, $Q' = \{n\}$, and δ consists of the transitions $(i, x, i+1)$, $(n, x, 1)$ where $i = 1, \ldots, n-1$ and $x \in \{a, b\}$. Then the size of A_n is $O(n \cdot \log n)$ and the number of primitive cycles in A_n is $O(2^n)$.*

By the above remarks, the decision algorithm given by the proof of Theorem 3.1 is extremely inefficient. However, it is useful because with small modifications the same method allows us to show that also the generalized (logarithmic or polynomial) fairness condition is decidable. This question was left open in [11]. First we consider the case where the fairness function is linear.

Theorem 3.2 *Let $g(x) = \tau x + \kappa$ be a linear function where τ and κ are constants. For a given context-free grammar G we can effectively decide whether or not $L(G)$ has the g-fairness property.*

Proof. Let Σ be the terminal alphabet of G. Similarly as in the proof of Theorem 3.1, by Propositions 3.1 and 3.2, it is sufficient to check, for a reduced finite automaton A, and for each pair of symbols $b, c \in \Sigma$, whether or not

$$(\forall w \in L(A)) \;\; | \, |w|_b - |w|_c \, | \leq \tau |w| + \kappa. \tag{4}$$

If τ is negative, we can decide (4) by determining whether $L(A)$ contains a word w that makes $\tau|w| + \kappa$ negative and then going through the finite number of shorter words. Thus, we can assume that τ is non-negative in the following.

We claim that (4) is equivalent to the below two conditions:

(i) For all accepting primitive paths α of A,

$$| \, |\mathrm{word}(\alpha)|_b - |\mathrm{word}(\alpha)|_c \, | \leq \tau |\mathrm{word}(\alpha)| + \kappa.$$

(ii) For all primitive cycles β of A,

$$| \, |\mathrm{word}(\beta)|_b - |\mathrm{word}(\beta)|_c \, | \leq \tau |\mathrm{word}(\beta)|.$$

First assume (4). This directly implies (i) since word(α) $\in L(A)$ for all accepting paths α. If (ii) would not hold then there exists a cycle β such that

$$| \, |\text{word}(\beta)|_b - |\text{word}(\beta)|_c \, | > \tau |\text{word}(\beta)|.$$

Since A is reduced, it follows that A has an accepting path of the form $\gamma_1 \cdot \beta \cdot \gamma_2$. (This is seen using a similar argument as in the proof of Theorem 3.1.) Since word($\gamma_1 \cdot \gamma_2$) $\in L(A)$, we have

$$| \, |\text{word}(\gamma_1 \cdot \gamma_2)|_b - |\text{word}(\gamma_1 \cdot \gamma_2)|_c \, | \leq \tau |\text{word}(\gamma_1 \cdot \gamma_2)| + \kappa.$$

Denote the constant $\tau|\text{word}(\gamma_1 \cdot \gamma_2)|$ by D, and let

$$\eta(m) = \gamma_1 \cdot \beta^m \cdot \gamma_2, \quad m \in \mathbb{N}.$$

Then

$$\text{word}(\eta(2D + 2\kappa + 1)) \in L(A)$$

violates the condition (4).

For the converse part assume the conditions (i) and (ii). Similarly as in the proof of Theorem 3.1, we observe that an arbitrary accepting path of A can be written in the form

$$\gamma_1 \cdot \eta_1 \cdot \gamma_2 \cdot \ldots \cdot \eta_{k-1} \cdot \gamma_k, \tag{5}$$

where $k \geq 1$, η_i, $1 \leq i \leq k-1$, is a cycle and $\gamma_1 \cdot \gamma_2 \cdots \gamma_k$ is a primitive accepting path. The inequality of condition (ii) extends to arbitrary cycles and, thus, the underlying word of (5) has to satisfy (4). □

We can naturally consider also the uniform linear fairness question: Given a context-free grammar G decide whether or not there exist constants τ and κ such that $L(G)$ has the $(\tau x + \kappa)$-fairness property. From the proof of Theorem 3.2 it follows that the answer to this question is "yes" iff all primitive cycles of the constructed automaton A contain occurrences of each symbol of Σ.

As a corollary we see that also the logarithmic and polynomial fairness conditions are decidable for context-free languages. The question of logarithmic or polynomial fairness is reduced, essentially, to the linear case by testing separately a finite number of special cases.

Corollary 3.1 *For a context-free language L it is decidable whether or not L has the* log*-fairness property.*

Proof. Let Σ be the alphabet of L. Exactly as in the first part of the proof of Theorem 3.2, in order to decide the log-fairness property for L, it is sufficient to decide for a reduced finite automaton A, and for all $b, c \in \Sigma$, whether or not

$$(\forall w \in L(A)) \;\; | \, |w|_b - |w|_c \, | \leq \log(|w|). \tag{6}$$

Let $b, c \in \Sigma$ be fixed. We first observe that if for some cycle α of A we have

$$|\text{word}(\alpha)|_b \neq |\text{word}(\alpha)|_c \tag{7}$$

then the condition (6) does not hold. To see this observe that since A is reduced, there exists an accepting path $\gamma_1 \cdot \alpha \cdot \gamma_2$ and, by pumping the cycle α sufficiently many times, we obtain an accepting path such that the corresponding underlying word violates (6).

We can effectively test that no cycle α satisfies (7) just by going through the primitive cycles. If this is the case, then we can determine whether or not (6) holds by checking the finite number of primitive accepting paths. □

Note that the above proof relies only on the fact that the fairness function grows asymptotically slower than any linear function.

Corollary 3.2 *Let $p : \mathbb{N} \to \mathbb{N}$ be a polynomial function. For a context-free language L over an alphabet Σ it is decidable whether or not L has the p-fairness property.*

Proof. Since the linear case was considered above we can assume that the rank of $p(x)$ is at least two. Furthermore, we may assume that the coefficient of the term of highest rank in $p(x)$ is positive because otherwise any infinite language would not have the $p(x)$-fairness property. (Note that we can effectively decide whether or not a given context-free language is infinite.)

Again it is sufficient to decide for a reduced finite automaton A, and for all $b, c \in \Sigma$, whether or not

$$(\forall w \in L(A)) \ \mid |w|_b - |w|_c \mid \leq p(|w|). \tag{8}$$

In the following let $b, c \in \Sigma$ be fixed. By our assumptions concerning the polynomial $p(x)$ we can effectively find $M \in \mathbb{N}$ such that

$$p(x) > x \ \text{ for all } x > M.$$

Since $\mid |w|_b - |w|_c \mid$ cannot be greater than $|w|$, in order to decide (8), it is sufficient to test the condition for words of $L(A)$ of length at most M. □

Intuitively, we can say that a super-linear fairness condition is satisfied unless it is violated by some word of constant length, where the constant depends only on the fairness function. It can be noted that the decision algorithm given by the above proof is extremely inefficient since it uses an exhaustive search over all words of at most a certain length.

Finally, we may note that [11] considered also a notion called *initial fairness*, where the fairness condition for symbols $b, c \in \Sigma$ is required to hold only "as long as" the remaining suffix contains occurrences of both symbols b and c. It was shown that the uniform initial contant-fairness question is decidable for context-free languages. The proof of Theorem 3.2 could fairly easily be modified to show that also the initial linear fairness (or polynomial fairness) condition is decidable. However, we feel that the notion of initial fairness is well motivated, perhaps, only in the constant case. The reason is that the standard constant fairness condition by itself is very restrictive as it requires that all words contain almost the same number of occurrences of arbitrary symbols b and c.

References

[1] J.-M. Autebert, J. Berstel and L. Boasson, Context-free languages and push-down automata, in: *Handbook of Formal Languages, Vol. I.* (G. Rozenberg, A. Salomaa, eds.), pp. 111–174, Springer-Verlag, 1997.

[2] H.-D. Burkhard, Fairness and control in multi-agent systems, *Theoret. Comput. Sci.* **189** (1997) 109–127.

[3] N. Francez, *Fairness,* Springer-Verlag, Berlin, 1986.

[4] M.A. Harrison, *Introduction to formal language theory,* Addison-Wesley, Reading, MA, 1978.

[5] C. Hartonas, A fixpoint approach to finite delay and fairness, *Theoret. Comput. Sci.* **198** (1998) 131–158.

[6] M. Kudlek and A. Mateescu, On distributed catenation, *Theoret. Comput. Sci.* **180** (1997) 341–352.

[7] W. Kuich, Semirings and formal power series, in: *Handbook of Formal Languages, Vol. I.* (G. Rozenberg, A. Salomaa, eds.), pp. 609–677, Springer-Verlag, 1997.

[8] A. Mateescu, CD grammar systems and trajectories, *Acta Cybernetica* **13** (1997) 141–157.

[9] A. Mateescu, G.D. Mateescu, G. Rozenberg and A. Salomaa, Shuffle-like operations on ω-words, manuscript 1996.

[10] A. Mateescu, G. Rozenberg and A. Salomaa, Shuffle on trajectories: Syntactic constraints, *Theoret. Comput. Sci.* **197** (1998) 1–56.

[11] A. Mateescu, K. Salomaa and S. Yu, Decidability of fairness for context-free languages, in: *Proceedings of the Third International Conference on Developments in Language Theory, DLT'97 (Thessaloniki, July 20–23, 1997),* S. Bozapalidis (ed.), pp. 351–364.

[12] R.J. Parikh, On context-free languages, *J. Assoc. Comput. Mach.* **13** (1966) 570–581.

[13] D. Park, Concurrency and automata on infinite sequences, in: "Theoretical Computer Science", *Proc. of the 5th GI Conference,* P. Deussen (ed.), Lect. Notes Comput. Sci. **104**, Springer-Verlag, (1981) 167–183.

[14] L. Priese, R. Rehrmann and U. Willecke-Klemme, An introduction to the regular theory of fairness, *Theoret. Comput. Sci.* **54** (1987) 139–163.

[15] J. Romijn and F. Vaandrager, A note on fairness in I/O automata, *Inform. Process. Lett.* **59** (1996) 245–250.

[16] A. Salomaa, *Formal languages,* Academic Press, New York, 1973.

[17] S. Yu, Regular languages, in: *Handbook of Formal Languages, Vol. I.* (G. Rozenberg, A. Salomaa, eds.), pp. 41–110, Springer-Verlag, 1997.

CURRICULUM VITAE

1. **Name:** Ferenc Gécseg

2. **Position:** Professor of Computer Science

3. **Institution:** Institute of Informatics, A. József University (Szeged, Árpád tér 2, Hungary)

4. **Personal data:**
   Date of birth: 13.3.1939,
   Place of birth: Zalavár (Hungary)
   Married since 1959 (Mária Pápai)
   Children: Mária (b. 1962) and Zsuzsanna (b. 1964)
   Grandchildren: Sándor (b. 1985), Andrea (b. 1989), Kató (b. 1990), Panna (b. 1991) and Szabolcs (b. 1994)

5. **Studies:**
   Elementary and general school in Zalavár (1945-1953)
   High school in Keszthely (1953-1957)
   University studies at the University of Szeged (1957-1962), graduated in 1962 as a mathematics teacher (with specialization in algebra)

6. **Degrees:**
   Candidate of Mathematical Sciences (1967, Hungarian Academy of Sciences)
   Doctor of Mathematical Sciences (1976, Hungarian Academy of Sciences)
   Corresponding Member of the Hungarian Academy of Sciences (1987)
   Full Member of the Hungarian Academy of Sciences (1995)

7. **Positions held in Hungary:**
   Professor's Assistant (1962-1965, A. József University)
   Aspirant (1965-1967, Hungarian Academy of Sciences)
   First Assistant to Professor (1968-1970, A. József University)
   Associate Professor (1970-1977, A. József University),
   Professor (1977-, A. József University)

8. **Visiting positions:**
   Postdoctorate Fellow, University of Manitoba, Canada (1.9.1968-31.8.1969)

Visiting Professor, University of Turku, Finland (1.9.1974- 31.8.1975)
Visiting Professor, Tampere University of Technology, Finland (1.9.1978-31.12.1978)
Visiting Professor, University of Western Ontario, Canada (1.1.1987-30.6.1987)
Visiting Research Professor, Academy of Finland (1.9.1992- 28.2.1993)

9. **Professional and association membership:**
Member of the Council of EATCS (since 1983)
Vice-President of EATCS (1989-1998)
Member of János Neumann Computer Science Society
Member of János Bolyai Mathematical Society
Editor of Acta Cybernetica
Editor of Alkalmazott Matematikai Lapok
Editor of Acta Mathematica Hungarica
Editor of Acta Scientiarum Mathematicarum
Editor of Foundations of Control Engineering

10. **Conferences:**
More than 30 invited talks at international conferences, summer schools and foreign scientific institutions.
Chairman of the mini-conferences on Algebraic Theory of Automata held in Szeged, in 1973 and 1977.
Chairman of the 1981 and 1989 international conferences on Fundamentals of Computation Theory.
Chairman of the Programme Committee of ICALP 95.
Program committee member for the following conferences:
- FCT 79 (Wendisch–Rietz, DDR, 1979)
- FCT 83 (Borgholm, Sweden, 1983)
- Algebra, Combinatorics and Logic in Computer Science (Győr, Hungary, 1983)
- FCT 85 (Cottbus, DDR, 1985)
- FCT 87 (Kazan, SSSR, 1987)
- FCT 93 (Szeged, Hungary, 1993)
- ICALP 94 (Lund, Sweden, 1994)
- MFCS 97 (Bratislava, Slovak Republic, 1997)

11. **Research interest:**
Automata and formal languages, universal algebra

12. Honorary degree and awards:

Foreign Member of the Finnish Academy of Sciences (1994)
Grünwald Géza Prize (1966)
Order of Labour (silver degree) (1974)
Academic Prize (1980)
Kalmár László Memorial Medal (1982)
Order of Labour (golden degree) (1983)
Szele Tibor Memorial Medal (1990)
Szent-Györgyi Albert Prize (1995)

List of Publications by Ferenc Gécseg

Books, Book Chapter, Editing

1. *Algebraic Theory of Automata,* Akadémiai Kiadó, Budapest, 1972 (with I. Peák).

2. *Tree Automata,* Akadémiai Kiadó, Budapest, 1984 (with M. Steinby).

3. *Products of Automata,* Springer-Verlag, Berlin–Heidelberg–New York–Tokyo, 1986.

4. *Fundamentals of Computation Theory,* Proceedings of the 1981 International FCT Conference, Szeged, Hungary, Springer Lecture Notes in Computer Science, Volume 117 (Editor).

5. *Fundamentals of Computation Theory,* Proceedings of the 1989 International FCT Conference, Szeged, Hungary, Springer Lecture Notes in Computer Science, Volume 380 (Editor with J. Csirik and J. Demetrovics).

6. *Automata, Languages and Programming,* 22nd International Colloquium, ICALP95, Szeged, Hungary, July 1995, Proceedings, Springer Lecture Notes in Computer Science (Editor with Z. Fülöp).

7. Tree languages, in: *Handbook of Formal Languages*, vol. 3, Springer-Verlag, Berlin–Heidelberg–New York–Tokyo, 1996, 1-69 (with M. Steinby).

Papers

1. Schreier extensions of multi-operator Ω-groups (in Russian), *Acta Sci. Math.,* **23** (1962), 58-63.

2. On some classes of semimoduls and moduls (in Russian), *Acta Sci. Math.,* **24** (1963), 165-172.

3. On products of ordered automata I (in Russian), *Acta Sci. Math.,* **24** (1963), 244-250.

4. On products of ordered automata II (in Russian), *Acta Sci. Math.,* **25** (1964), 124-128.

5. On the mathematical theory of automata (in Hungarian), *Year Book of MTESZ* Szeged, 1964, 45-63 (with I. Peák).

6. Automata with isomorphic semigroups (in Russian), *Acta Sci. Math.*, **26** (1965), 43-47 (with I. Peák).

7. On loop-free compositions of automata (in Russian), *Acta Sci. Math.*, **26** (1965), 264-272.

8. On the groups of one-to-one mappings defined by finite automata (in Russian), *Kibernetika (Kiev)*, 1965 N°1, 37.

9. On the groups of automaton permutations (in Russian), *Kibernetika (Kiev)*, 1965 N°5, 14-17 (with B. Csákány).

10. Algebraic theory of automata (in Hungarian), *Mat. Lapok*, **17** (1966), 77-134 (with I. Peák).

11. On R-products of automata I, *Studia Sci. Math. Hungar.*, **1** (1966), 437-448.

12. On R-products of automata II, *Studi Sci. Math. Hungar.*, **1** (1966), 443-447.

13. On R-products of automata III, *Studia Sci. Math. Hungar.*, **2** (1967), 163-166.

14. On the family of automaton mappings (in Russian), *Acta Sci. Math.*, **28** (1967), 39-54.

15. On many-tact automata (in Russian), *Acta Sci. Math.*, **28** (1967), 55-63.

16. Complete systems of automata, *Proceedings of the 2nd Czechoslovak Conference on Automata Theory*, Brno, 1968, 59-62.

17. Metrically complete systems of automata (in Russian), *Kibernetika (Kiev)*, 1968 N°3, 96-101.

18. On complete systems of automata, *Acta Sci. Math.*, **30** (1969), 295-300.

19. On certain classes of Σ-structures, *Acta Sci. Math.*, **31** (1970), 191-195.

20. On equational classes of unoids, *Acta Sci. Math.*, **34** (1973), 99-101 (with S. Székely).

21. Model theoretical methods in the theory of automata, *Proceedings of the Symposium on Mathematical Foundations of Computer Science*, High Tatras, 1973, 57-63.

22. Fundamentals of automata theory (in Hungarian), *Természet Világa*, **104** (1973), 232-238.

23. On subdirect representation of finite commutative unoids, *Acta Sci. Math.*, **36** (1974), 33-38.

24. Composition of automata, *Proceedings of the 2nd Colloquium on Automata, Languages and Programming,* Saarbrücken, 1974, Springer Lecture Notes in Computer Science, Volume 14, 351-363.

25. On loop-free composition of commutative automata, *Proceedings of the Symposium on Discrete Systems,* Riga, 1974, 128-137.

26. Isomorphic representation of automata, *Proceedings of the 4th Symposium on Mathematical Foundations of Computer Science,* Marianske Lazne, 1975, Springer Lecture Notes in Computer Science, Volume 32, 226-230.

27. Representation of automaton mappings in finite length, *Acta Cybernet.,* **2** (1976), 285-289.

28. On products of abstract automata, *Acta Sci. Math.,* **38** (1976), 21-43.

29. On representation of trees and context-free languages by tree automata, *Found. Control Engrg.,* **1** (1976), 161-168 (with Gy. Horváth).

30. Universal algebras and tree automata, *Fundamentals of Computation Theory,* Proc. 1977 International FCT Conference, Poznan-Kornik, Springer Lecture Notes in Computer Science, Volume 53, 98-112.

31. Algebra and logic in theoretical computer science, *Mathematical Foundations of Computer Science,* Proc. 6th Symp., Tatranská Lomnica 1977, Springer Lecture Notes in Computer Science, Volume 54, 78-92 (with P. Tóth).

32. Minimal ascending tree automata, *Acta Cybernet.,* **4** (1978), 37-44 (with M. Steinby).

33. Algebraic theory of tree automata I (in Hungarian), *Mat. Lapok,* **26** (1975), 169-207 (with M. Steinby).

34. Algebraic theory of tree automata II (in Hungarian), *Mat. Lapok,* **27** (1976-1979), 283-336 (with M. Steinby).

35. On the periodic sum of finite automata, *Found. Control Engrg.,* **5** (1980), 229-231 (with B. Imreh).

36. Tree transformations preserving recognizability, in: *Finite Algebra and Multiple-Valued Logic,* Szeged, 1979, Coll. Math. Soc. J. Bolyai, Volume 28, North-Holland P.C., 1981, 251-273.

37. On complete systems of tree automata, *Conference on System Theoretical Aspects in Computer Science,* Salgótarján, 1982, 103-111.

38. On a representation of deterministic frontier-to-root tree transformations, *Acta Sci. Math.,* **45** (1983), 177-187.

39. On a representation of deterministic uniform root-to-frontier tree transformations, *Acta Cybernet.,* **6** (1983), 173-180.

40. General products and equational classes of automata, *Acta Cybernet.*, **6** (1983), 281-284 (with Z. Ésik).

41. Products of automata, *Proceedings of the Summer School on Applications of Mathematics in Techniques,* Varna, 1984. 5-14.

42. Finite representations and infinite products, *Papers on Automata Theory,* 1984-2, 55-63.

43. Metric representations by ν_i-products, *Acta Cybernet.*, **7** (1985), 203-209.

44. On ν_i-products of commutative automata, *Acta Cybernet.*, **7** (1985), 55-59.

45. Metric equivalence of tree automata, *Acta Sci. Math.*, **48** (1985), 163-171.

46. Type-independent equational classes and metric equivalence of tree automata, *Fund. Inform.*, **9** (1986), 205-216 (with Z. Ésik).

47. On α_i-products of automata. Homomorphic representation, in: *Algebra, Combinatorics and Logic in Computer Science,* Győr, 1983, Coll. Math. Soc. J. Bolyai, Volume 42, North-Holland P.C., 1986, 403-421.

48. Homomorphic realization of automata with composition, *Proc. Symp. on Mathematical Foundations of Computer Science,* Bratislava, 1986, Springer Lecture Notes in Computer Science, Volume 233, 299-307 (with P. Dömösi, Z. Ésik, and J. Virágh).

49. On α_0-products and α_2-products, *Theoret. Comput. Sci.*, **48** (1986) 1-8 (with Z. Ésik).

50. On metric equivalence of ν_i-products, *Acta Cybernet.*, **8** (1987), 129-134 (with B. Imreh).

51. On α_i-products of tree automata, *Acta Cybernet.*, **8** (1987), 135-141 (with B. Imreh).

52. On a representation of tree automata, *Theoret. Comput. Sci.*, **53** (1987), 243-255 (with Z. Ésik).

53. A comparison of α_i-products and ν_i-products, *Found. Control Engrg.*, **12** (1987), 3-9 (with B. Imreh).

54. Automata over algebras with dimension, *Proceedings of the East European Category Seminar,* Predele, 1987, 11 (with H. Jürgensen).

55. Characterizations of locally transitive semiautomata, *Papers on Automata Theory,* 1987-2, 1-8 (with G. Thierrin).

56. On a special class of tree automata, *2nd Conf. on Automata, Languages and Programming Systems,* 1988. 141-152 (with B. Imreh).

57. The role of theory in computer science, *EATCS Bulletin,* Number 30, 1988, 295-297.

58. A decidability result for homomorphic representation of automata by α_0-products, *Acta Math. Hungar.,* **53** (1-2) (1989) 205-212 (with Z. Ésik).

59. On star products of automata, *Acta Cybernet.,* **9** (1989), 167-172 (with B. Imreh).

60. Simulation by ν_1^*-products of automata, *Publ. Math.,* **36** (1989), 51-56 (with P. Dömösi).

61. Automata represented by products of soliton automata, *Theoret. Comput. Sci.,* **74** (1990), 163-181 (with H. Jürgensen).

62. On $\alpha_0 - \nu_1$-products of automata, *Theoret. Comput. Sci.,* **80** (1991), 35-51 (with H. Jürgensen).

63. Simulation and representation by ν_i^*-products of automata, *Publ. Math.,* **40** (1992), 75-83 (with P. Dömösi).

64. A note on isomorphically complete systems, *Discrete Appl. Math.,* **36** (1992), 307-311 (with B. Imreh).

65. Algebras with dimension, *Algebra Universalis,* **30** (1993), 422-446 (with H. Jürgensen).

66. On finite isomorphically complete systems of tree automata, *Acta Sci. Math.,* **57** (1993), 497-502 (with B. Imreh).

67. On homomorphic representations by products of tree automata, *Results and Trends in Theoretical Computer Science,* Proc. Symp., Graz 1994, Springer Lecture Notes in Computer Science, Volume 812, 131-139.

68. On completeness of nondeterministic automata, *Acta Math. Hungar.,* **68** (1995), 151-159 (with B. Imreh).

69. On the cube-product of nondeterministic automata, *Acta Sci. Math.,* **60** (1995), 321-327 (with B. Imreh).

70. On two classes of formal languages (in Hungarian), *Polygon,* **5** 1995, 1-13.

71. Dependence in algebras, *Fund. Inform.,* **25** (1996), 247-256 (with H. Jürgensen).

72. On complete sets of tree automata, Proceedings of the 3rd International Conference *Developments in Language Theory,* Thessaloniki, July 20-23, 1997, 37-47 (with B. Imreh).

73. On the existence of finite isomorphically complete systems, *J. Aut. Lang. and Comb.,* to appear (with B. Imreh and A. Pluhár).

74. On quasi-products of tree automata, *Discrete Appl. Math.*, (submitted for publication).

75. On some classes of tree automata and tree languages, *Ann. Acad. Sci. Fenn. Math.*, (submitted for publication).

CONTENTS

Sponsored by SYSDATA Ltd.

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János
A kézirat a nyomdába érkezett: 1999. február
Terjedelem: 13,7 (B/5) ív